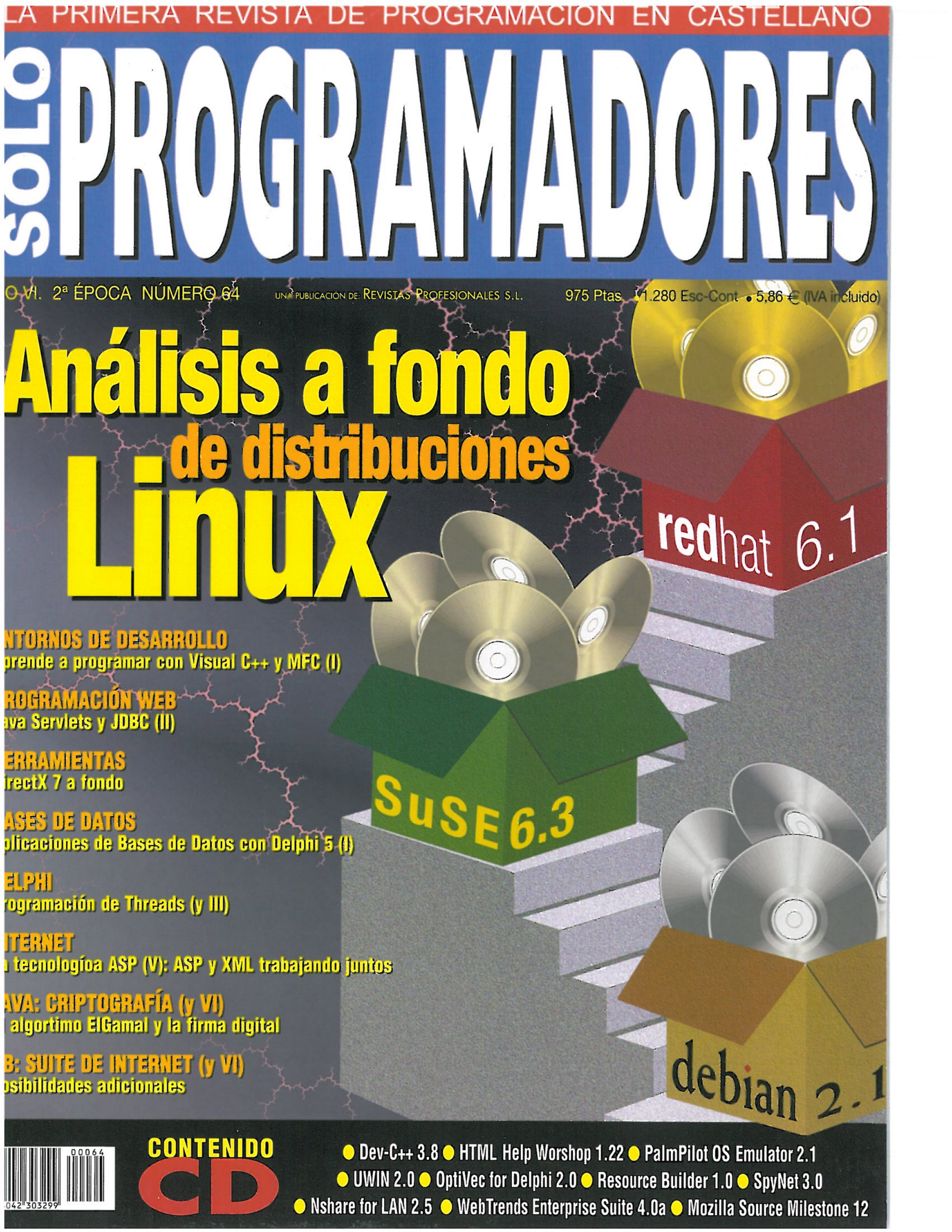

LA PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO
SOLO PROGRAMADORES
O VI. 2ª ÉPOCA NÚMERO 64
UNA PUBLICACIÓN DE: REVISTAS PROFESIONALES S.L.
975 Ptas • 1.280 Esc-Cont • 5,86 € (IVA incluido)
Análisis a fondo
de distribuciones
Linux
redhat 6.1
SuSE 6.3
debian 2.1
NTORNOS DE DESARROLLO
prende a programar con Visual C++ y MFC (I)
ROGRAMACIÓN WEB
ava Servlets y JDBC (II)
ERRAMIENTAS
irectX 7 a fondo
ASES DE DATOS
plicaciones de Bases de Datos con Delphi 5 (I)
ELPHI
rogramación de Threads (y III)
TERNET
a tecnologíoa ASP (V): ASP y XML trabajando juntos
AVA: CRIPTOGRAFÍA (y VI)
algortimo ElGamal y la firma digital
B: SUITE DE INTERNET (y VI)
osibilidades adicionales
CONTENIDO
CD
Dev-C++ 3.8
HTML Help Worshop 1.22
PalmPilot OS Emulator 2.1
UWIN 2.0
OptiVec for Delphi 2.0
Resource Builder 1.0
SpyNet 3.0
Nshare for LAN 2.5
WebTrends Enterprise Suite 4.0a
Mozilla Source Milestone 12
00064
042 303299

SÓLO PROGRAMADORES

Número 64
SÓLO PROGRAMADORES
es una publicación de
REVISTAS PROFESIONALES S.L.

Editor
Agustín G. Buelta
Director
Javier Amado Buiza
Coordinador Técnico
Eduardo De Riquer Frutos
Coordinadoras de Redacción
Gema Romero Moreno-Manzanaro
Cristina Peña del Pozo
Colaboradores
Constantino Sánchez, Juan Luis Ceada,
Javier Sanz, Adolfo Aladro,
Enrique de la Lastra,
Jordi Agost, Vicente A. Sánchez Werner
Ilustración de portada
Manuel Gómez Allende
Maquetación y Tratamiento de Imagen
Paco Risco
Consultas técnicas
atecnica@virtualsw.es

Publicidad
Tel.: (91) 304 78 46
Mariano Sánchez (Barcelona)
Tel.: (93) 322 12 38
Pepín Gallardo (Barcelona)
Tel.: 617 09 36 68

Suscripciones
Rosa Tabares
Tel. (91) 304 87 64 Fax: (91) 327 13 03

Preimpresión
Grebe
Impresión
I. de Impresión
Distribución
Motorpress Ibérica

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 30-4-2000
Precio en Canarias, Ceuta y Melilla:
938 ptas. sin I.V.A.
Con sobretasa aérea: 975 ptas. sin I.V.A.

EDITORIAL

Programación independiente

Linux es un sistema operativo que no se debe dejar de lado. Muchos son los lenguajes de programación que son independientes, en lo que a plataformas se refiere, y que permiten desarrollar tanto bajo *Windows* como bajo *Linux*. Por ello, y haciendo caso de las sugerencias de nuestros lectores, hemos querido realizar un análisis de las tres distribuciones más conocidas que según la mayoría de usuarios son las mejores.

En este podium encontramos *RedHat* 6.1, *SuSE* 6.3 y *Debian* 2.1. Seguro que los que no conocéis el mundillo *Linux* os quedaréis sorprendidos por algunas de las características de estas nuevas versiones. Y es que este sistema operativo ha tenido un verdadero boom en los 90 y a comienzos del 2000 parece ser que se confirman todavía más sus halagüeñas expectativas.

No hace mucho tiempo se han puesto en la Red, a disposición de todos aquellos usuarios que deseen descargárselas, las archiconocidas librerías *DirectX* 7. Por lo que aquí os contamos las nuevas características y prestaciones que ofrecen. A destacar que con varias de ellas se reduce la carga de trabajo sobre el procesador agilizando el resto de procesos.

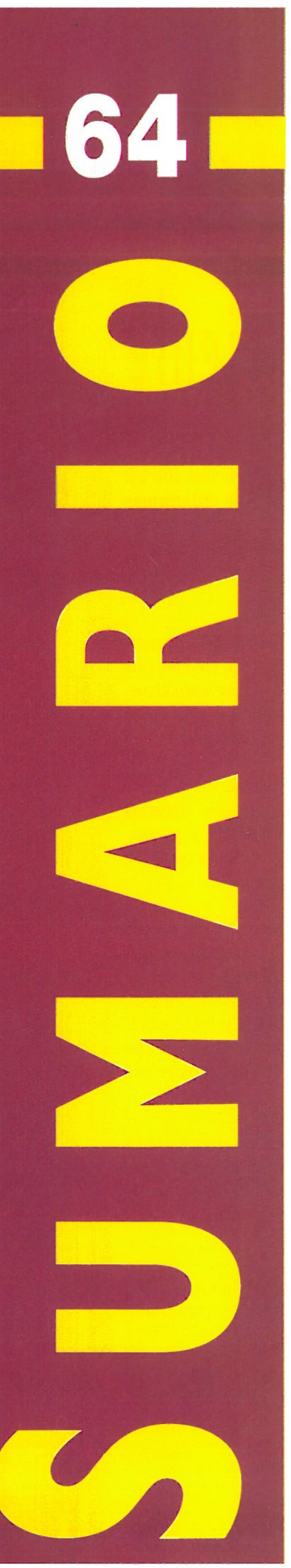
Otro de los artículos que nos habéis solicitado con insistencia es la creación de bases de datos, especialmente con *Delphi*. Por ello comenzamos una serie donde hablamos de todo esto y mucho más. Para aquellos que no dispongan de muchos conocimientos sobre el tema que no se asusten, en esta primera parte comenzamos con explicaciones muy generales y los conocimientos mínimos necesarios no son muy altos. Aunque a lo largo de las distintas partes nos iremos adentrando más y profundizando en aquellos temas que creemos de mayor relevancia.

En nuestra sección de Internet tratamos uno de los lenguajes más en boga en *Internet*, *XML*, eso sí, sin olvidar que estamos trabajando con *ASP*. Una asociación que permite a los creadores de páginas experimentados y a los programadores exprimir aún más las posibilidades de trabajo en el servidor directamente.

En este número finalizan varios reportajes, pero no os preocupeis, porque para el siguiente número estamos preparando novedades que seguro que os agradarán.

Hasta el número que viene.

Javier Amado
Director



SÓLO PROGRAMADORES

46 Bases de Datos

APLICACIONES DE BASES DE DATOS EN DELPHI 5 (I)

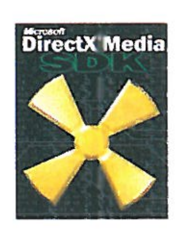
Comenzamos este mes una nueva serie de artículos dedicados al desarrollo de aplicaciones que trabajan con diferentes bases de datos. En cualquier caso, nos centraremos sobre todo en aquellas que ofrece *Delphi* 5. A lo largo de éste y de los próximos números abordaremos diversos temas, desde las herramientas a los componenetes BDE hasta los conceptos más avanzados pasando por la utilización de bases de datos remotas.



38 Herramientas

MICROSOFT DIRECTX 7

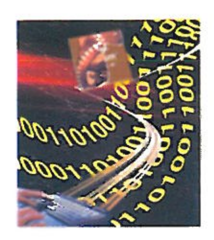
Recientemente ha llegado al usuario la última revisión de *DirectX* (la 7), de tal forma que éste es un momento propicio para indagar en las nuevas características que se han incorporado, que no son pocas...



54 Delphi

PROGRAMACIÓN DE THREADS (y III)

Para finalizar con esta serie, en esta entrega veremos cómo acceder a bases de datos desde múltiples hilos, cómo optimizar la creación de *threads* y la forma de trabajar directamente con la *API* de *Windows*.



60 Internet

LA TECNOLOGÍA ASP (V): ASP y XML TRABAJANDO JUNTOS

XML ha irrumpido con fuerza en *Internet*, por el momento el único navegador que soporta el estándar es *Internet Explorer 5.0*. Sin embargo, *XML* puede ser utilizado en el lado del servidor, dentro de las páginas *ASP*. Esto nos permite aprovechar las ventajas de *XML* con independencia del navegador.



66 Java

CRIPTOGRAFÍA (y VI)

En este último artículo se completará la descripción de los algoritmos de clave pública con la presentación del algoritmo *ElGamal*. Además, se mostrarán los conceptos de funciones unidireccionales y firma digital, que actualmente son necesarios para realizar la mayoría de las operaciones de comercio electrónico en *Internet*.

72 Visual Basic

SUITE DE INTERNET (y VI): POSIBILIDADES ADICIONALES

¿Cómo podemos arrancar automáticamente una conexión a *Internet*? ¿Cómo podemos utilizar *Outlook* como gestor de correo? ¿De qué forma podemos saber el estado de todos los puertos de comunicación? A estas y otras preguntas intentaremos responder en el último artículo de esta serie.



78 Libros

Las últimas novedades sobre programación y los libros que nos han parecido más interesantes aparecen en estas páginas, con una sencilla reseña que os ayude a conocerlos mejor y a decidir cuál es el que más os conviene.

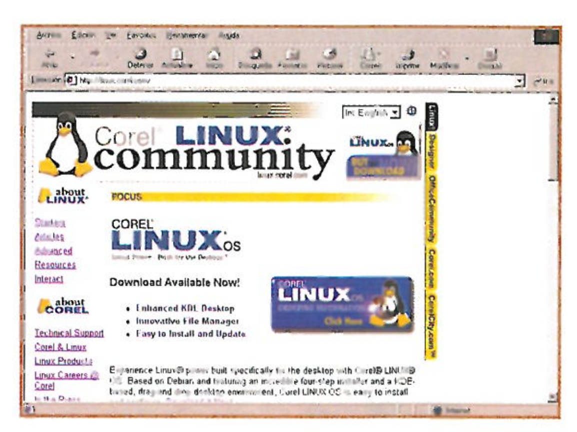
80 Dudas Técnicas

Como siempre en esta sección contestamos a todas vuestras dudas. No os preocupéis si os parecen demasiado complejas o demasiado sencillas, siempre podréis acudir a nuestra dirección de *E-mail*: solop@virtualsw.es. Nosotros intentaremos solucionar vuestros problemas.



COREL LINUX PODRÁ ACCEDER A APLICACIONES WINDOWS Y EJECUTARLAS

Corel Corporation ha anunciado que su escritorio *Corel Linux* será el primer sistema operativo que ejecute perfectamente aplicaciones *Windows* a través de cualquier conexión. La habilidad de acceder a aplicaciones *Windows* y utilizarlas convierte a *Corel Linux* en la solución ideal para los usuarios de *PC* que deseen acelerar su transición al escritorio *Linux*, permitiéndoles utilizar al completo sus aplicaciones *Windows*.



Gracias a la integración de *GraphOn Bridges* en el sistema *Corel Linux* será fácil disfrutar de las ventajas y de la alta fiabilidad de *Linux* mientras se tiene acceso a las aplicaciones *Windows* más utilizadas, según declaraciones de los directivos de la empresa. Este paso tecnológico permitirá a las organizaciones hacer uso de los escritorios *Linux* y *Windows* en el mismo *PC* sin complicaciones, uno de los principales objetivos que se ha marcado *Corel*.

Hay que resaltar que el software *GraphOn Bridges* permite a cualquier dispositivo de visualización ejecutar toda clase de aplicación a través de cualquier tipo de conexión, entre las que figuran conexiones de banda estrecha, telefónica e inalámbrica. También es capaz de que cualquier aplicación (ya sea *Windows, Linux* o *UNIX*) sea ejecutada de manera instantánea a través de *Web*, sin necesidad de hacer ninguna modificación de *software*.

Está previsto que una versión de este sistema con cliente *Linux* y licencias de servidor Windows NT para *GraphOn Bridges* se comercialice a mediados del año 2000.

Los usuarios podrán acceder desde lugares remotos a la mayoría de las aplicaciones *Windows* en un servidor *Windows NT* a través del escritorio *Corel Linux*, sin tener que adquirir licencias de *software* de otros fabricantes. Por ejemplo, un grupo de *PC* que ejecuten el sistema *Corel Linux* pueden a su vez ejecutar aplicaciones *Linux* nativas tales como las próximas *suites* de *Corel* (ofimática o gráfica) para *Linux*, y también ejecutar aplicaciones *Windows* como *Microsoft Office,* desde un único servidor basado en *Windows* compartido por el grupo *Linux*.

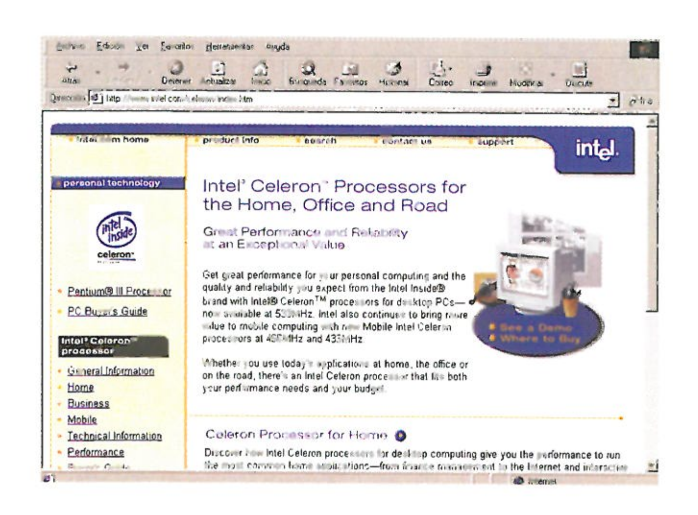
Para más información: **linux.corel.com**

NUEVO PROCESADOR CELERON A 533 MHZ

La empresa *Intel Corporation* da a conocer su nuevo procesador *Intel Celeron* a 533 *MHz*, su procesador más rápido para los *PC's* con un precio inferior a los 1.000 dólares.

Este novedoso procesador mejora las prestaciones globales de los *PC's* ofrecidos en el segmento de mercado de los ordenadores asequibles y aporta a los usuarios buenas prestaciones a un precio excelente en relación con su calidad. Consigue ofrecer un mejor rendimiento frente a las anteriores versiones más lentas existentes hasta el momento en el mercado.

Para más información: **www.intel.com**

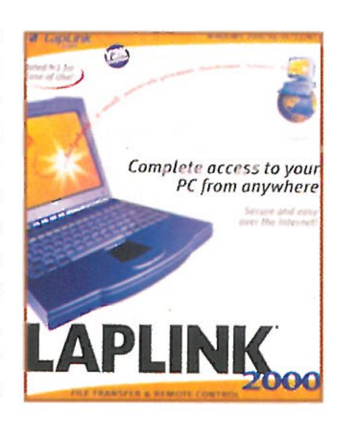


LAPLINK.COM PRESENTA LAPLINK 2000

LapLink.com, ha anunciado la próxima disponibilidad de *LapLink* 2000 en castellano, la actualización de su producto estrella *LaplLink* que permite a los usuarios acceso fácil y seguro así como mover datos entre dos *PC's* vía *Internet*.

Esta nueva utilidad se presenta totalmente compatible con *Windows* 2000. El producto le saca el máximo partido a *Internet* para conectar dos *PC's* de forma rápida y segura. Además de su control remoto y transmisión de archivos, *LapLink* 2000 incluye una gama completa de nuevas funciones de seguridad como por ejemplo: encriptación de doble nivel, sofisticado mecanismo de seguridad, contraseñas con caracteres en minúsculas y mayúsculas, etc.

Una de las funciones más interesantes de esta herramienta es la de *Link-ToNet* que deja a un lado las limitaciones de las conexiones estándar de *módem* a *módem*, de este modo no se permite al usuario remoto el acceso completo a los recursos de la red, incluyendo *E-mail, Internet*, unidades de red e impresoras.



Para más información: **www.laplink.com**

LA INDUSTRIA ESPAÑOLA APOYA EL LANZAMIENTO DE WINDOWS 2000

La industria española al completo, incluyendo los fabricantes de *PC's*, de *software*, vendedores de *hardware* e integradores de sistemas, ratifican la preparación de *Windows* 2000 para entornos de empresa, gracias a las especiales características de las que ha sido dotado para su uso en *Internet*, su fiabilidad, su escalabilidad, su elevado nivel de seguridad y la reducción que supone del coste de propiedad en entornos de empresa.

Dicha plataforma, sucesora de *Windows NT* 4.0, supone la base para la próxima generación de sistemas de información; además de ello presenta la ventaja de poder ser usada tanto en portátiles, como ordenadores y servidores en *cluster*. El nuevo sistema operativo ofrece a las compañías una mayor facilidad en el uso de *Internet*, de un modo seguro y con una infraestructura manejable que permite una optimización del *hardware*.

Microsoft Windows 2000 ofrece un mejor funcionamiento de las aplicaciones y está diseñado para mejorar la arquitectura informática y la disponibilidad de los negocios 24 horas al día, ofreciendo una plataforma más segura. Además de ello facilita la manejabilidad de la siguiente generación de ordenadores permitiendo a los clientes realizar una mejor utilización de *Internet*.

Para más información: **www.microsoft.com**

NUEVA ARQUITECTURA DE 12000 TERABIT SYSTEM DE CISCO

Terabit System 12000 de *Cisco* ya ha sido presentado en sociedad; se trata de un nuevo producto de esta compañía cuya arquitectura proporciona una mayor escalabilidad a las infraestructuras *IP* de los proveedores de servicios.

El nuevo 12000 *Terabit System* llega a los 5 *Terabits* por segundo utilizando una avanzada arquitectura de fábrica de conmutación y ofrece una mayor escalabilidad a los *backbones IP* de las operadoras, de modo que puedan hacer frente a la creciente demanda de tráfico. Debido a que el sistema es gestionado desde un único nodo de enrutamiento, la arquitectura *POP* se simplifica y se reducen a los costes operativos.

La arquitectura de esta nueva herramienta es única entre las plataformas de enrutamiento finales, basándose en una estructura sólida y ampliamente instalada.

FUJITSU ENTRA EN LINUX

Fujitsu está convencida de las ventajas que ofrecen los nuevos sistemas operativos, especialmente la emergente alternativa *Linux* para informática personal, por lo que proporcionar *software* en este terreno es un paso lógico a seguir.

Partiendo de esta idea se presenta en el mercado *TeamWARE Office* 5.3 para *Linux*, un producto que verá la luz en el mercado japonés pero que después realizará su lanzamiento global.

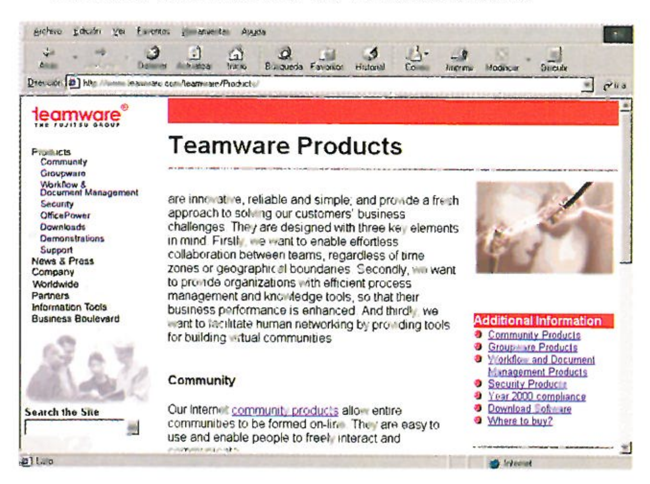
Esta nueva herramienta supone una solución de *intranet* ideal para pymes que necesitan soporte independiente para costes y ventas bajo estándares comunes industriales. Se trata de un conjunto de aplicaciones de trabajo en grupo que proporcionará a los usuarios una forma sencilla de realizar sus tareas administrativas y la comunicación con los demás.

TeamWARE Office 5.3 incluye prestaciones para correo electrónico, planificación del tiempo y recursos, debates, gestión y recuperación de documentos, facilita las tareas de búsqueda, visualización, análisis y actualización de la información de una forma rápida.

Entre sus características destacan su preparación para ejecutar *groupware* pudiéndose construir inmediatamente una infraestructura de comunicación en *Linux* para los usuarios a través de los *browsers* de la *Web* e interfaz estándar.

La edición 2 de *TeamWARE* Office 5.3 puede ser soportada en *Caldera Open Linux, Red Hat* y *Turbo Linux*.

Para más información: www.teamware.com



NETWORK ASSOCIATES RECIBE PERMISO PARA LA EXPORTACION DE SU SOFTWARE DE CIFRADO PGP ENCRYPTON



Network Associates ha conseguido la concesión por parte del gobierno de Estados Unidos de una licencia total para exportar su producto *PGP*, líder en el mercado de *software* de cifrado. Esto marca el fin de una vieja prohibición, vigente durante décadas, sobre la exportación de productos de cifrado de alto nivel.

La licencia, de efectos inmediatos, permite a esta compañía exportar su producto de *software* de cifrado *PGP* a casi todos los países del mundo sin restricciones. Existe un amplio consenso sobre la necesidad de productos de cifrado de alto nivel disponibles en todo el mundo que permitan realizar transacciones seguras en los negocios electrónicos a través de *Internet*.

Network Associates ofrece hoy en día su tecnología de cifrado y autentificación de *PGP* en varios productos, dirigidos tanto a empresas como a particulares.

Recientemente dicha empresa fue nombrada como principal proveedor mundial de *software* de cifrado, así como el mayor proveedor de *software* de seguridad en general.

Para más información: **www.nai.com**

HERRAMIENTAS DE PROGRAMACIÓN



ENTORNOS DE PROGRAMACIÓN

DEV-C++ 3.8

Editor multiventana para la creación de programas orientados a proyectos. Incorpora un compilador de *C* y *C++* y un depurador, lo que agiliza las tareas de compilar, enlazar y ejecutar las aplicaciones que están siendo creadas. También cuenta con una utilidad para crear programas de instalación, numerosos "include" y más de 100 librerías.

JPADPRO 3.7 BUILD 326

Entorno de desarrollo integrado para generar aplicaciones *Java* y editar ficheros *HTML*. Sus opciones permiten la detección de errores de sintaxis, alternar las herramientas de desarrollo *Java* de *Sun* y *Microsoft*, localizar clases, métodos y programas, e incluso es posible desarrollar nuevas herramientas usando *JavaScript* o *VBScript*.

MICROSOFT HTML HELP WORKSHOP 1.21

Crea ficheros *HLP* y páginas *Web* que utilizan controles navegables al estilo de los ficheros de ayuda de *Windows* 98. Incluye un gestor de proyectos, un editor de imágenes, un control *ActiveX*, un *applets Java* y un compilador que consigue ficheros de tamaño reducido.

PALMPILOT OS EMULATOR 2.1

Emula el *hardware* de varios modelos de dispositivos de *Palm Computing Platform*. Con este emulador es posible utilizar un dispositivo virtual *Palm* en el escritorio del ordenador. Resulta muy útil para escribir, probar y depurar aplicaciones, especialmente si se utiliza junto con las utilidades *Gremlins* y *Debug Palm OS ROM's*.

TEXTURIZER 1.71

Completo editor de textos capaz de sustituir al *Notepad* de *Windows*. Puede abrir varios documentos en una misma ventana y situarlos como iconos en la barra de tareas. Soporta macros, cliptext (listas de cadenas de textos que pueden ser insertadas en los documentos), menús contextuales, y exporta e importa ficheros de *Unix* y *Macintosh*.

UWIN 2.0

Ofrece un entorno de desarrollo y ejecución *UNIX* para programadores de *Windows*. Permite crear aplicaciones *Unix* en la plataforma *Windows*, de modo que puedan migrar entre plataformas volviendo a compilar el código fuente. Incluye utilidades *Unix* y *Ksh* para crear *scripts*, accesos remotos a clientes y servidores, utilidades y editores *Unix*. Además incorpora librerías de cliente *X11R6* y soporte de desarrollo de aplicaciones *X11* para *Windows*.

LENGUAJES

VISUAL BASIC

ACTIVEZIPPER PRO 1.0F BUILD 1098

Pequeño y rápido control de compresión y descompresión de ficheros *ZIP* para *Visual Basic*. Utiliza el formato *PKZIP*, puede encriptar ficheros y archivos, protegerlos por *password*, etc. Realiza verificaciones *CRC* para detectar errores en la transmisión de datos.

VISUAL BASIC SPLASH ANIMATOR 5.0

Genera animaciones para su ejecución en aplicaciones de *Visual Basic*. Sólo hay que crear o seleccionar las imágenes, los movimientos y pulsar un botón para que se genere automáticamente todo el código fuente, las imágenes se compilen en un fichero de recursos listo para su ejecución en una aplicación Visual Basic.

VBSYS TRAY 1.00.12

Sitúa automáticamente un icono de un programa *VB* en la barra de tareas de *Windows*. Se trata de una solución sencilla que añade un control a un formulario *Visual Basic* y establece sus propiedades. Para permitir a una aplicación responder mientras está escondida, *VBSysTray* soporta eventos de clic, doble clic, ratón arriba y abajo, como respuesta a la selección del icono por el usuario.

JAVA

FORTE FOR JAVA COMMUNITY EDITION

Entorno de trabajo para *Java* que facilita el desarrollo de aplicaciones. Incluye el núcleo del entorno de desarrollo integrado junto con una colección de módulos diseñados para agilizar y mejorar el proceso de desarrollo. Es personalizable y cuenta con una estructura modular que puede ser ampliada con *plugins*.

JAVA SHARED DATA TOOLKIT 2.0

Librería de desarrollo que añade funciones de colaboración a applets y aplicaciones escritas en *Java*. Puede ser utilizada para crear aplicaciones de red como entornos de *chat*, paneles de información, presentaciones

remotas y para facilitar la distribución de datos que agilicen el trabajo en equipo. Es compatible con las tecnologías multimedia de *Sun.*

PERSONALJAVA EMULATION ENVIRONMENT 3.02

Entorno de desarrollo para verificar que *applets PersonalJava* y las aplicaciones que se han creado utilizando *Java Development Kit* y funcionan en el entorno de aplicaciones *PersonalJava.*

HTML

ARACHNOPHILIA 4.0 BUILD 5268

Editor *HTML* que permite visualizar las páginas internamente o utilizando 6 navegadores diferentes. Soporta *tags* para insertar elementos *HTML* avanzados. Puede convertir documentos realizados en un procesador de texto a *HTML.* Incluye un cliente *FTP,* un verificador de errores, soporte para macros y tutoriales de *HTML, JavaScript* e *Internet.*

HYPERTEXT BUILDER 8.0

Editor *HTML* que combina la comodidad de las herramientas *WYSIWYG* con la potencia de la edición de código fuente. Incluye una herramienta de transferencia *FTP,* un editor de mapas de imágenes, diálogos para insertar variables de servidores *ASP, applets Java, scripts,* etc.

OTROS

3D ACTIVE MULTIMEDIA BUTTON 4.5

Control *ActiveX* que crea atractivos botones utilizando distintas formas, superficies, efectos *3D, bitmaps,* cursores, animaciones, sonidos, colores y fuentes. Es compatible con *Microsoft FrontPage* y *Microsoft Internet Explorer.*

@CTIVIDEO ACTIVEX CONTROL 1.4

Incorpora a un programa opciones para realizar capturas de imágenes, vídeo y editarlas. Puede ser utilizado en entornos que soporten *ActiveX* como *Visual Basic, Visual C++, Delphi 3.0* y programas *MS Office.*

FLIPPER CAD CONTROL 2.05

Control *ActiveX* que añade funciones de *CAD* a los programas. Puede ser utilizado en la mayoría de los entornos de desarrollo *OCX* de 32 *bits,* incluyendo *Microsoft Access, Delphi, Visual Basic,* y *Visual Foxpro.* Incluye 16 herramientas de dibujo, numerosas tramas para las líneas, el relleno y opciones de color.

INFOPOWER 2000 COMPONENT SUITE FOR DELPHI 5

Colección de componentes para distribuir aplicaciones de base de datos. Entre otros componentes cuenta con filtros visuales, controles de calendarios, un procesador de texto *RichEdit* y un analizador de datos.

METAKIT 2.0

Librería de clases *C++* para desarrolladores que necesiten guardar cualquier tipo de información estructurada de una manera sencilla pero segura. Utiliza una sencilla *API* basada en 6 clases para gestionar el almacenamiento de los datos.

MOZILLA SOURCE MILESTONE 12

Código fuente en *C/C++* de *Mozilla.* Nueva actualización de este popular navegador de *Internet.* El código fuente está cubierto por la licencia pública de *Netscape.* Esta versión sólo se puede utilizar para su aprendizaje.

OPTIVEC FOR DELPHI 2.0

Estamos tratando con una colección de más de 3.000 funciones matemáticas para *Delphi.* Ofrecen mayor rapidez que el código fuente *Delphi* compilado para las mismas funciones.

HERRAMIENTAS Y UTILIDADES

007 WRITE ALL STORED PWL PASSWORDS (WASP) 2.0

Muestra todos los passwords que se almacenan en el fichero *PWL* de Microsoft y ofrece la posibilidad de borrar este fichero. El programa sólo visualiza los *passwords* correspondientes al usuario que ha abierto la sesión.

ADVANCED ECOMMERCE BUILDER 1.0

Aplicación que facilita el mantenimiento y diseño de una tienda virtual. Permite utilizar la base de datos del servidor, las páginas *ASP* y un diseño intuitivo para presentar un negocio electrónico.

RESOURCE BUILDER 1.0

Potente herramienta para construir de forma visual *scripts* de controles de recursos y ficheros de recursos. Amplía las funciones de *Borland Resource WorkShop* y *Borland Image Editor.* Ofrece *bitmaps,* opciones para personalizar iconos, cursores y colores, editores para los tipos de recursos más comunes como menús diálogos, cursores, iconos, etc. Soporta ficheros *JPEG.*

SOURCE CONTROL 1.1

Ayuda a la gestión del código fuente de los proyectos con una interfaz visual que muestra el árbol de directorios. Proporciona las versiones de todos los ficheros modificados para poder volver a cualquier de ellas si es necesario.

SPYNET 3.0

Utilidad de monitorización de redes que permite a los administradores identificar transmisiones no autorizadas, los datos capturados, posibles puntos débiles de la red y *passwords* que han sido desencriptados. Soporta *Windows NT* y servidores *proxy.*



REDES

ATELIER WEB TCP PORT SCANNER 1.10

Escáner de puertos que permite escanear rápidamente múltiples puertos de una red. Genera informes de las conexiones y los datos recibidos incluyendo los números de los puertos locales y remotos, el número del *socket* local y la descripción de los servicios estándar de un puerto remoto determinado.

INTERNET MONITOR 2.02

Monitoriza la mayoría de los aspectos de una red de un proveedor de servicios de *Internet (ISP)*. Puede probar los siguientes servicios: *DNS, FTP, E-mail (POP3 y SMTP), News (NNTP), WINS* y *WWW*, y verificar conexiones *TCP/IP* orientadas a servicios como *MS SQL* o *Telnet*.

NETWORK ASSISTANT 1.9

Utilidad de comunicación de redes que permite comunicar con otros usuarios de una *LAN* en tiempo real. Ofrece tres modos de comunicación: conversación, panel y servicios especiales para todos los usuarios, sólo un grupo o un usuario.

NSHARE FOR LAN 2.5

Ofrece acceso simultáneo a la red a un grupo de ordenadores. Para ello hay que instalar el componente del servidor y los clientes que se desee. Utiliza módems *DSL*, Cable y analógicos y soporta numerosos programas como *Internet Exchange, Netscape, Real Audio, Electronic Mail, Telnet, FTP*, etc.

SOHOCCONNECTION 2.0

Aplicación *Java* que permite a los ordenadores de una red compartir una conexión a *Internet*. La conexión se puede realizar a través de un módem convencional, dispositivos *ISDN/RDSI, cable-modem* o *ADSL*. El *gateway* de *SOHOConnection* también trabaja como un firewall, protegiendo la red de los intrusos. Soporta cualquier plataforma *Java*.

VICOMSOFT SOFTROUTER PLUS 6.52

Potente programa para conectar una red *LAN* a *Internet*, redes *LAN* a *LAN* y redes *LAN* a *WAN*. Facilita la conexión de la red a *Internet* y, con una única cuenta *ISP*, los usuarios pueden consultar sitios *Web* o utilizar otros servicios internos de red como el correo electrónico.

WEBTRENDS ENTERPRISE SUITE 4.0A

Colección de herramientas que comprueban el funcionamiento de sitios *Web* y las conexiones con *Internet*. Ofrece la posibilidad de enviar los resultados de los análisis a servidores de bases de datos como *Oracle 7/8, Microsoft SQL, Sybase* e *Informix*.

DOCUMENTACIÓN/ TUTORIALES

HTML HELPER 1.0

Tutorial de *HTML* que ayuda a comprender los fundamentos de este lenguaje. Incluye el código básico para crear sencillas páginas y, en otra guía, ofrece el código más avanzado.

LEARNING FRONTPAGE 2000 V1.0

Completo manual de creación de páginas *Web* utilizando *FrontPage 2000*. Abarca todos los elementos necesarios para el desarrollo de las páginas *HTML*.

LEARNING PHOTOSHOP 5.5

Curso de creación de imágenes para *Internet* con *Adobe PhotoShop* 5.5. Comienza con una introducción acerca del programa y luego desarrolla los fundamentos.

THINKING IN JAVA

Versión en *HTML* del libro de programación "*Thinking in Java*" de *Bruce Eckel*. El libro ofrece claras explicaciones, numerosos ejemplos y ejercicios de programación.

IMPRESCINDIBLES

ANTIVIRUS

AntiViral Toolkit Pro 3.0.129
AVP Virus Enciclopedia
McAfee VirusScan 4.0.3
Panda Antivirus 6.13.00 Platinum

GRÁFICOS

Icon Bank 4.0 Gold Edition
IconForge 4.6
MicroAngelo 98 v4.77
Paint Shop Pro with Animation Shop 6.01
Reptile 2.0
SureThing CD Labeler 2.0
ThumbsPlus 4.02
Ultra Fractal 2.04
Xara WebStyle 1.2

INTERNET

Añadir Pro 4.01.003
AutoWinNet 6.0 Beta 1
Copernic 2000 v4.01
Cuentapasos 3.77
CuteFTP 3.5.6
Dial-Up Magic 1.8
Eudora Light 3.0.6
Free Agent 1.21
GetRight 4.1.2
Go!Zilla 3.5
Guardián 1.1
HomeSite 4.01
ICQ 99b beta 3.19 build 2569
i.Share 3.5
MIRC 32 5.61
URL Organizer 2.3.2
WebZIP 3.06
WinGate 3.0.5

MULTIMEDIA

AudioCatalyst 2.01
CDH Media Wizard 4.12
COWON Jet-Audio 4.7
MusicMatch Jukebox 4.4
Sonique 1.30
WinAMP 2.50e

NAVEGADORES

Internet Explorer 5.01
NeoPlanet 5.0.0.1057
Netscape Communicator 4.7
Opera 3.61

PROGRAMACIÓN

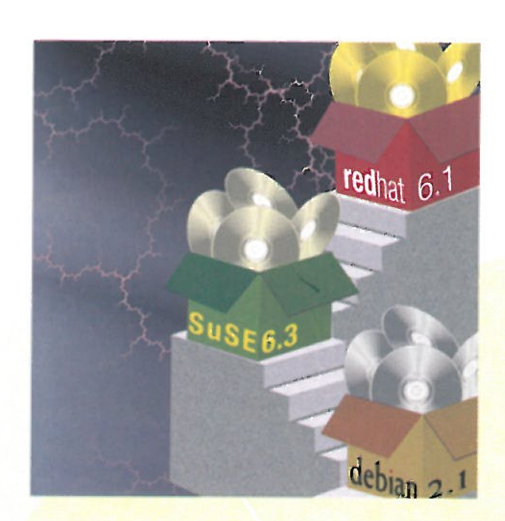
Decafe Pro 3.6
Free Pascal v0.99.12
Hackman 4.02
Help & Manual 2.25
InstallConstruct 3.3
Java Development Kit 1.2.2-001
UltraEdit Professional
Text/HEX Editor 7.00a
Windows Registry Guide 1.3
WinHex 8.85

UTILIDADES

Adobe AcrobatReader 4.0
Advanced Registry Tracer (RegFix) 1.11
Babylon Translator Spanish 2.1.46
CallCenter 3.5.8
Calendar Builder 3.2j
DirectX 7.0
Emergency Recovery System 9.03
Nero 4.0.7.5
SiSoft Sandra 99.8.5.30
Where Is It? 2.15a
Windows Commander 4.03
WinZip 7.0

ATENCIÓN:
En caso de problemas con el CD-ROM envíelo por correo ordinario, a la atención del SERVICIO TÉCNICO DE SÓLO P., incluyendo en el interior del sobre sus datos personales, a la siguiente dirección:
C/ San Sotero, Nº 5, 1ª Planta, 28037 (Madrid)

Distribuciones Linux para Programadores



Vicente A. Sánchez Werner.
Desarrollador independiente.

Uno de los mayores atractivos que posee *Linux* de cara a los usuarios es la posibilidad de elegir entre diversas distribuciones, aunque para los desarrolladores esto ha sido siempre una fuente de problemas. Para que todos los conceptos queden claros, en este artículo analizaremos las tres principales distribuciones del momento: *RedHat 6.1, SuSE 6.3* y *Debian 2.1.*

DISTRIBUCIONES LINUX Y LOS ESTÁNDARES

En este artículo veremos qué influencia tienen, en nuestro trabajo como desarrolladores, la existencia de diversas distribuciones, así como las características más relevantes que pueden influir en los programadores y en su utilización como plataformas de desarrollo.

Para poder ver el impacto que tiene la existencia de diversas distribuciones *Linux* debemos remontarnos brevemente al origen de este sistema operativo. Creado a comienzos de la década de los 90, *Linux* se desarrolló como una alternativa gratuita a otro sistema operativo ya existente, *Minix*. Esto motivó que para acelerar el desarrollo del sistema y facilitar su acceso se optase por portar las herramientas del proyecto *GNU*, obligando a que el sistema siguiese siendo gratuito.

En este punto es cuando aparecen en escena las distribuciones, que son recopilaciones de *software* con el objetivo de crear un sistema plenamente funcional, a diferencia de lo que se obtendría teniendo sólo una parte, como el *kernel* o las librerías. El nacimiento de las distribuciones marca la aparición de *Linux* como sistema operativo de pleno derecho y ha sido el motor que ha impulsado el desarrollo de los nuevos instaladores gráficos y la difusión de *Linux* entre un público cada vez más numeroso.

Estas distribuciones eran creadas inicialmente por una o más personas que decidían qué *software* debería incluirse y su organización. Como resultado aparecieron distintas distribuciones organizadas de maneras diferentes y con *software* que variaba de ligeramente entre cada una de ellas, llevando de forma inevitable a la aparición de ligeras incompatibilidades.

Linux se presenta en forma de recopilaciones de software llamadas distribuciones

Esta situación se fue agravando con la llegada de nuevas distribuciones y la aparición de los primeros sistemas de paquetes (*.RPM* y *.DEB*) haciéndose cada vez más evidente la necesidad de algún estándar que pusiese un poco de orden en este revuelto mundo.

Todo esto llevó a la aparición del *LFHS*, o *Linux Filesystem Hierarchy Standard*, que definía una estructura común de directorios y sus contenidos para todas las distribuciones, rebajando en gran medida el nivel de incompatibilidad derivado de la existencia de diversos archivos en directorios diferentes a los esperados por las aplicaciones.

Este estándar es aceptado por las distribuciones actuales, aunque no siempre se cumple estrictamente, abundando interpretaciones distintas que provocan ligeras incompatibilidades.

Pese a todos estos esfuerzos, el nivel de incompatibilidad entre diferentes distribuciones es bastante elevado cuando consideramos que todas son expresiones del mismo sistema operativo, siendo especialmente frecuente cuando nuestros programas usan librerías dinámicas. Esta situación ha llevado a un nuevo esfuerzo de estandarización, conocido como *Linux Standard Base* o *LSB*.

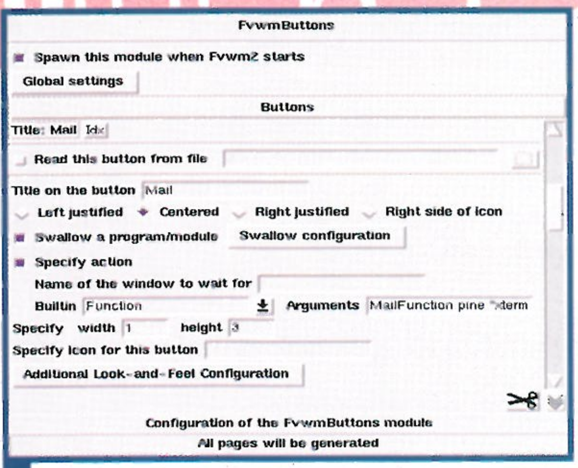


Figura 1.- Pantalla de configuración del *Fwm* 95 de *Debian*.

Este esfuerzo tiene como principal objetivo el crear un estándar a todos los niveles que permita que los desarrolladores de *software* comercial puedan crear un programa que funcione a la perfección en las diversas distribuciones sin necesidad de adaptaciones específicas. Sin embargo, este esfuerzo aún se encuentra en un estado inicial de creación de especificaciones y no ha generado ningún resultado plenamente visible.

Actualmente las distribuciones *Linux* representan la base sobre la que debemos crear programas, y pese a la existencia de estándares, su poca implantación no soluciona todavía nuestros problemas, por lo que generalmente deberemos analizar qué distribución nos interesa para un determinado tipo de desarrollo según sus prestaciones, facilidad de uso e instalación, estabilidad y el coste económico de la misma y el derivado de su uso.

DISTRIBUCIONES LINUX, LAS TRES GRANDES

En el mundo *Linux* siempre han existido diversas tendencias, intereses y filosofías que han terminado cristalizando en las numerosas distribuciones que existen en el mercado, cada una con su particular filosofía e idiosincrasia. Esta fragmentación ha creado y destruido gigantes y, hoy por hoy, la mayor parte de los usuarios de *Linux* se inclinan por una de las siguientes distribuciones: *Debian 2.1*, *RedHat 6.1* y *SuSE 6.3*.

Las diferentes distribuciones existentes representan las diversas tendencias entre los usuarios de Linux

Aparte de las tres ya citadas existe una legión de ellas, derivadas o de nueva creación, que se están haciendo un hueco en el mundo *Linux*, pero generalmente no presentan novedades importantes sobre las tres mencionadas anteriormente.

DEBIAN 2.1 (SLINK)

La distribución *Debian 2.1r4*, también conocida como *Slink*,

es la última versión aparecida de esta famosa y reputada distribución. Ha sido creada por un numeroso grupo de voluntarios, partidarios todos ellos del *software Open Source* y con el objetivo de obtener un sistema libre y altamente estable.

Debian está considerada como la distribución más estable en el mundo Linux

Debian es considerada por muchos la distribución para desarrolladores por excelencia, quizás por el énfasis que sus creadores ponen en la consecución de un sistema lo más estable y seguro posible así como en el fuerte carácter multiplataforma de esta distribución. Actualmente *Debian* es la distribución que está disponible en un mayor número de plataformas y arquitecturas.

La instalación de *Debian* es a ciencia cierta su mayor defecto, máxime si la comparamos con los nuevos instaladores gráficos que incorporan las distribuciones más modernas.

Este proceso se efectúa en dos pasos diferentes, uno llevado a cabo mediante una interfaz en modo texto y que permite realizar la configuración básica del sistema, así como la instalación del mismo. Este proceso es el más sencillo, pero debemos tener en cuenta que al terminarlo, no tendremos un sistema plenamente funcional, sino una mera base que permite arrancar y realizar la segunda parte de la instalación.

La segunda parte de la instalación de *Debian* consiste en dar una clave de superusuario, elegir la combinación de paquetes que deseamos instalar en nuestra máquina y realizarla. El primer paso es trivial, permitiéndonos crear un usuario adicional para que realicemos tareas habituales con él.

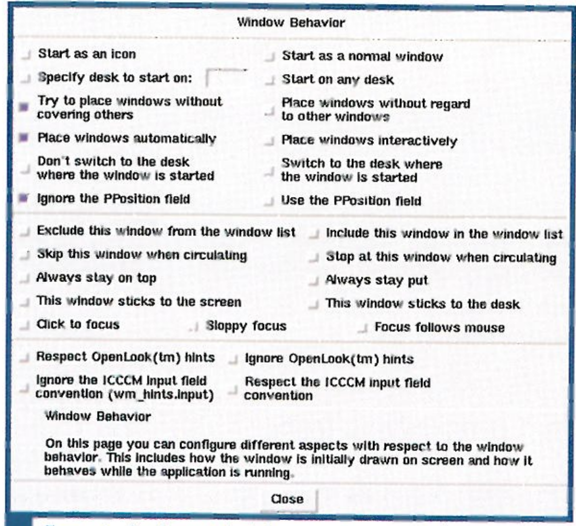


Figura 2.- Configuración del comportamiento de las ventanas en *Debian*.

El segundo paso ha mejorado considerablemente desde versiones anteriores de *Debian* y ahora podemos realizar una selección de *software* basándonos en perfiles predefinidos o creando un perfil propio incluyendo en él las tareas para las que vamos a utilizar el sistema.

A partir de este punto se abandona el sistema de instalación que estábamos usando y se pasa al antiguo sistema, comandado por el programa *dselect*. Es en este punto donde muchos usuarios inexpertos comenzarán a quejarse, ya que pese a aportar una tremenda flexibilidad, *dselect* es muy hostil al usuario, con un manejo excesivamente complicado.

La instalación de Debian es su mayor deficiencia: lenta, incómoda y anticuada en comparación con los de otras distribuciones

En cualquier caso, al seleccionar los diversos perfiles de *software,* el procedimiento de instalación nos informará sobre cómo realizar la instalación utilizando tres opciones y evitando la necesidad de aprender este complicado programa. Sin embargo debemos tener cuidado en no desviarnos de este procedimiento con el problema que supondría que se perdiese la lista de paquetes a instalar.

Si hemos seguido correctamente el procedimiento, el sistema comenzará la instalación de los programas seleccionados, pero no podremos desentendernos de este proceso porque se requiere nuestra intervención para realizar diversas elecciones o pasar algunos parámetros para que se configuren adecuadamente los programas, parándose

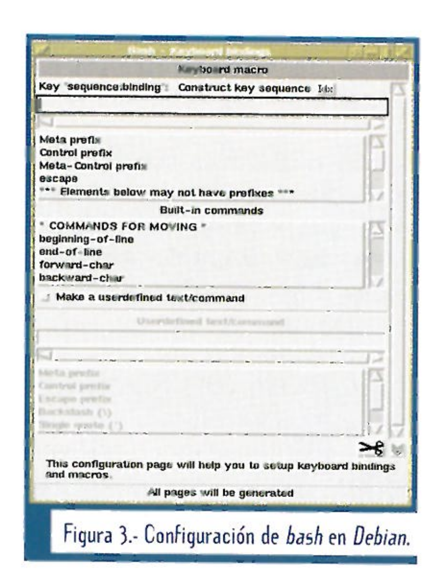
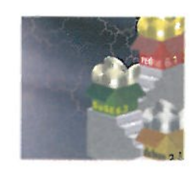
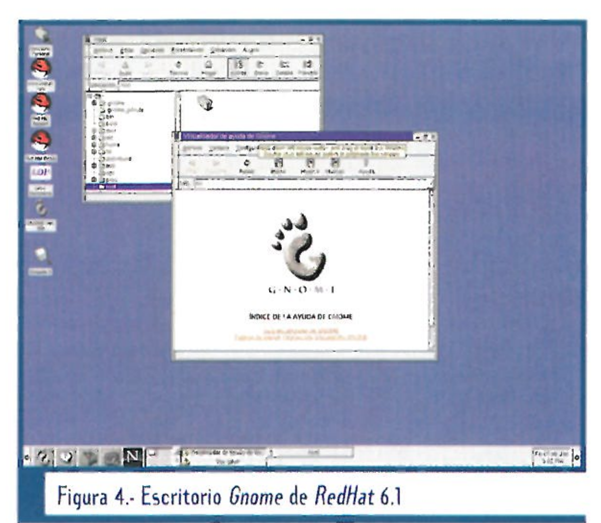


Figura 3.- Configuración de *bash* en *Debian*.




Figura 4.- Escritorio *Gnome* de *RedHat* 6.1

por completo mientras no realicemos dicha elección. Este pesado proceso hace que la instalación se prolongue mucho más que en otras distribuciones, lo que puede ser un freno de cara a los desarrolladores que por unos motivos u otros han de cambiar de máquinas frecuentemente, realizar la instalación en el local de un cliente, etc.

Debian es una buena opción si nuestro objetivo es el desarrollo de software libre

En estos momentos *Debian* está desarrollando un nuevo programa de instalación que sustituirá al anticuado *dselect* y que permita una instalación más eficiente y cómoda, lo que no está reñido con la flexibilidad y potencia, tal y como lo demuestran los nuevos instaladores gráficos desarrollados por otras distribuciones.

De cara al desarrollador, *Debian 2.1* ofrece una plataforma muy estable basada sobre un *kernel 2.0.38* y las librerías *glibc 2.0*, una combinación algo desfasada teniendo en cuenta que las demás distribuciones han migrado a *kernels* de la serie *2.2.x* y usan las librerías *glibc 2.1* (la propia *Debian* usa estas librerías en sus ediciones para plataformas no *Intel*).

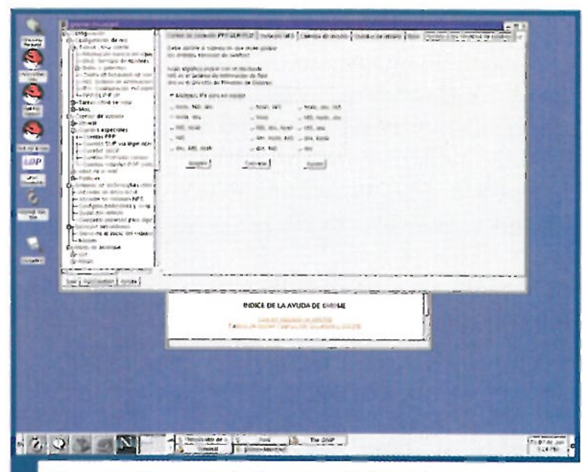
Esta elección representa, más que un retraso tecnológico, el extremo cuidado que pone *Debian* en la consecución de un sistema estable, no abandonando tecnologías de probada estabilidad hasta que no ha comprobado fehacientemente esta característica en una nueva combinación.

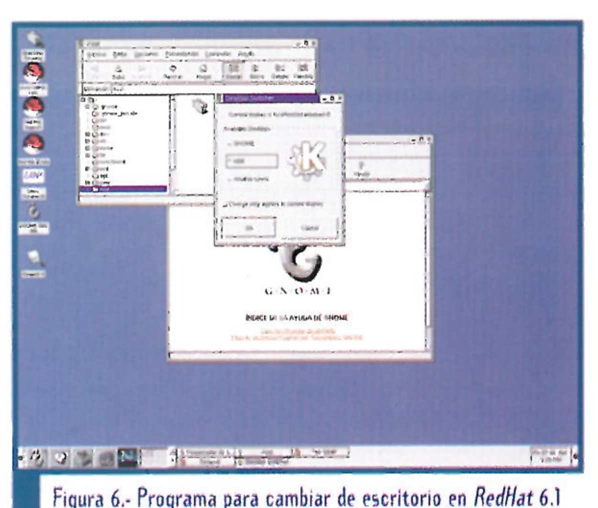
Las herramientas que acompañan a *Debian* son numerosas y una buena porción de ellas están dedicadas al desarrollo, como librerías, compiladores, lenguajes, etc. Sin embargo, en *Debian* encontramos algunas notorias ausencias derivadas de la política de no incluir *software* libre (según los estándares de *Debian*), como el escritorio *KDE* o las librerías *QT*, aunque siempre podemos bajarlas de la Red. También podemos encontrar otras sonoras ausencias como la del escritorio *Gnome,* quizás sólo debidas al exceso de cautela que tiene *Debian* a la hora de incorporar nuevos paquetes de *software*.

En cuanto al desarrollo, *Debian* constituye una excelente plataforma para el desarrollo de aplicaciones por diversas razones. La primera es la gran cantidad de aplicaciones para el desarrollo que se incluyen en la distribución, entre las que encontramos la *suite* de compiladores *GNU*, las últimas versiones aparecidas de los lenguajes *Perl* y *Python*, así como gran cantidad de útiles librerías para la programación, como *libjpeg*, *libpng* y similares, probadas al límite, por lo que podemos confiar en su estabilidad.

Un último punto que se debe considerar es que la propia filosofía de *Debian* ha provocado que algunas compañías que producen *software* para el desarrollo de aplicaciones no ofrezcan versiones específicas para esta distribución. En este caso lo más habitual es que la aplicación funcione tras instalar las librerías adecuadas, pero su funcionamiento puede no ser óptimo, y cualquier programa desarrollado de esta forma requerirá la instalación de dichas librerías. Además de que el soporte que nos dará la compañía será mínimo, debido a que se está usando su herramienta en un entorno no soportado oficialmente.

Teniendo en cuenta estos factores, *Debian* es una distribución especialmente indicada para el desarrollo de *software* libre o que va a funcionar en un entorno considerado "libre" por esta distribución, o *software* que requiera un entorno extremadamente estable a cualquier precio.


Figura 5.- *linuxconf* bajo *Gnome* en *RedHat* 6.1


Figura 6.- Programa para cambiar de escritorio en *RedHat* 6.1

En el otro extremo, la naturaleza de *Debian* restringe aún más el desarrollo, ya que debemos tener en cuenta que no es una distribución apta para principiantes o usuarios inexpertos, razón por la que no tiene mucho sentido tomarla como base para la creación de aplicaciones para usuarios finales.

REDHAT 6.1

La distribución *RedHat* es posiblemente la más conocida de todas las distribuciones de *Linux*, pese a no ser la más antigua. La última versión de esta distribución es la *6.1*, que está disponible en varias versiones (*Standard, Deluxe* y *Professional)* y para tres arquitecturas diferentes (*Intel, Sparc* y *Alpha*). La más habitual y la más difundida es la de libre distribución (*Standard*), que será la que tomaremos como referencia en este artículo.

RedHat está producida por la empresa del mismo nombre, pionera en el campo de las distribuciones *Linux*. Esta empresa ofrece servicios a empresas comparables a los ofrecidos por otros fabricantes de sistemas operativos, como soporte telefónico, servicios de formación, mantenimiento etc. Además *RedHat* colabora activamente en el desarrollo de *Linux*, pagando a programadores independientes o cediendo programadores a proyectos importantes para la comunidad *Linux*.

RedHat siempre ha estado enfocada a la obtención de un sistema cómodo de instalar y administrar a la vez que siempre ha intentado estar a la última, lo que le ha costado ser calificada de inestable y de contener fallos con demasiada frecuencia. Esta calificación no es del todo cierta, aunque sí lo es el que posee algunos problemas menores, y más cuando tomamos en cuenta la renombrada "estabilidad" de cierto sistema operativo comercial de todos conocido.

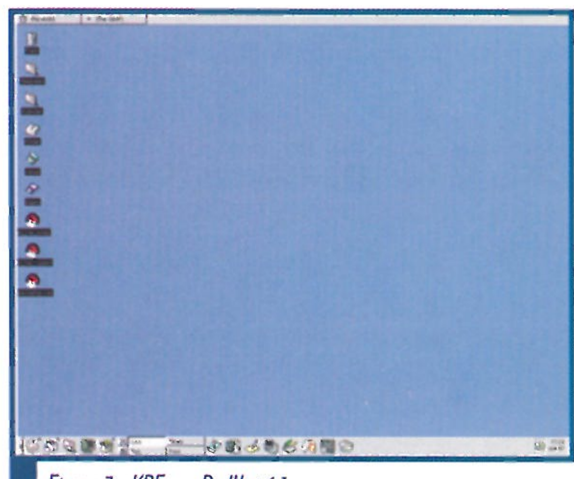
El hecho de que exista un mayor nivel de fallos que en *Debian*, o el que sea ligeramente más inestable, no es motivo para desechar esta distribución, que ofrece a cambio unas ventajas muy importantes de cara al desarrollador y al usuario final.

La actual distribución de *RedHat* está basada en *Glibc 2.1* y un *kernel 2.2.12*, pudiéndose considerar como mucho más avanzada en ese aspecto que *Debian*, e incorpora una novedad muy importante respecto a versiones anteriores y que puede ser de especial relevancia de cara a un desarrollador: la inclusión de un sistema de instalación gráfico, sencillo y cómodo que pone al alcance de cualquiera la instalación de este producto. Una instalación sencilla y un entorno cómodo que permiten ampliar la base de usuarios y facilitan la creación de soluciones *Linux* para empresas cuyos empleados no están familiarizados con el entorno.

RedHat está disponible en tres versiones diferentes

El procedimiento de instalación es muy sencillo y agradable, además de contar con una ayuda en todo momento (eso sí, en inglés) que explica exactamente y de forma clara en qué paso estamos, qué debemos hacer y cuáles son las recomendaciones en caso de existir diversas opciones. Este proceso de instalación hace muy sencilla la transición de *Windows* a *Linux*, aunque todavía *RedHat* no haya llegado al extremo de sencillez de este sistema operativo o de *Corel Linux*.

El instalador permite escoger el idioma que vamos a querer utilizar durante el procedimiento de instalación y en el sistema operativo, por lo que exceptuando la ayuda (aun en inglés), el resto del progra-


Figura 7.- *KDE* en *RedHat* 6.1.

ma aparecerá en nuestra lengua (con algún que otro gazapo o anglicismo suelto). Así, a diferencia de *Debian*, tenemos un sistema cómodo y sencillo para realizar la instalación de nuestro sistema operativo.

Una vez realizada la instalación ya disponemos de un sistema plenamente funcional, aunque en ocasiones habrá que ejecutar la utilidad *setup* para afinar un poco más la configuración del sistema y eliminar rápidamente aquellos pequeños fallos de configuración que hayamos podido crear con nuestra acción, o hayan sido motivados por un fallo del sistema de instalación. Esta facilidad de instalación y configuración es una gran ventaja de cara a desarrollar productos para usuarios finales, porque acerca a éstos al sistema y permite que en muchas ocasiones puedan resolver ellos mismos pequeños problemas que puedan aparecer, además de rebajar el tiempo necesario para instalar soluciones informáticas basadas en *Linux*.

La distribución está claramente orientada al trabajo en entornos gráficos, especialmente usando *Gnome*, aunque se incluye también el *KDE 1.1.2*, así como otros escritorios y manejadores de ventanas para poder adaptar al máximo la forma de trabajo con los gustos y/o necesidades del usuario. Esto permite una gran flexibilidad al programador ya que puede plantear el desarrollo para la integración en uno u otro de los escritorios incorporados, con la seguridad de que siempre va a tener acceso a las librerías de todos sin necesidad de bajarlas de *Internet*.

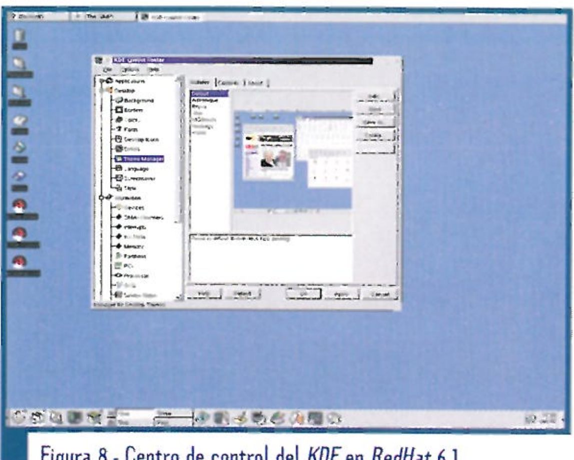


Figura 8.- Centro de control del *KDE* en *RedHat* 6.1.

En el ámbito de la programación y el desarrollo de aplicaciones, *RedHat 6.1* viene muy bien provista de compiladores y librerías, pero sin llegar a la cantidad de las incorporadas por *Debian*. En el campo de los compiladores encontramos los habituales de la *suite GNU*, en las últimas versiones, así como los diferentes entornos para *Perl* y *Python*.

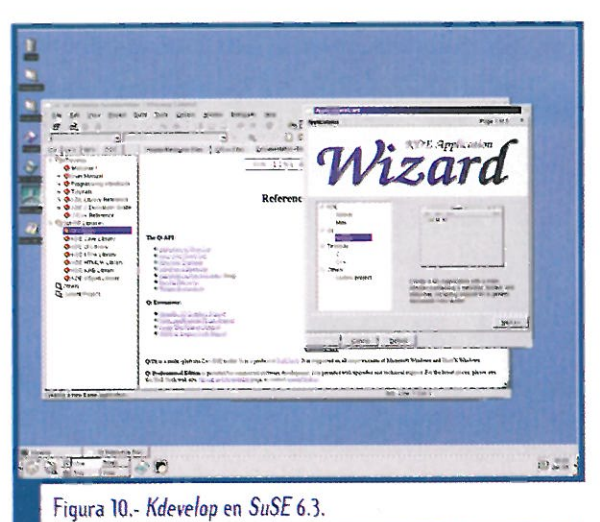


Figura 10.- *Kdevelop* en *SuSE* 6.3.

Una de las mayores bazas de *RedHat* es que gracias a su política de alianzas ha conseguido ser la primera en contar con aplicaciones de desarrollo comerciales, como *MetroWerks CodeWarrior*, aunque, a posteriori, se hayan desarrollado versiones adaptadas para algunas otras distribuciones como *SuSE*. Este hecho es de primera importancia de cara a las empresas de desarrollo de aplicaciones, ya que el poder contar con una herramienta respaldada comercialmente, aporta tranquilidad y seguridad para las instancias de la empresa encargadas de tomar las decisiones más comerciales (aunque eso no implica que las herramientas *GNU* estén a la misma altura o que no existan empresas que den un soporte de calidad para estas herramientas).

Dicho esto, cabe destacar que *RedHat*, en un movimiento más hacia el futuro, ha optado por incluir la versión *1.1.2* de *EGCS*, que es el compilador que está sustituyendo al tradicional *gcc*, por las diversas mejoras que aporta en cuanto a la compilación de código en *C++*.

Estos motivos y la ya mencionada política de alianzas, así como el peso específico adquirido recientemente en la bolsa, hacen que *RedHat* sea una empresa fuerte que ofrece un producto tecnológicamente más avanzado que *Debian*, y con la posibilidad de encontrar productos comerciales adaptados a la filosofía y funcionamiento específico de la distribución.

Estas ya serían razones más que suficientes para considerar esta distribución como un buen producto en el que basar nuestros desarrollos, pero el hecho de contar con un procedimiento de instalación y actualización sencillo y asequible y un entorno de trabajo gráfico óptimo, hacen que podamos tomarlo como base para la creación de aplicaciones asequibles a todo el mundo. De esta forma, *RedHat* coloca su distribución en una situación que hace un año escaso era casi imposible: la sustitución completa de *Windows* en el entorno empresarial.

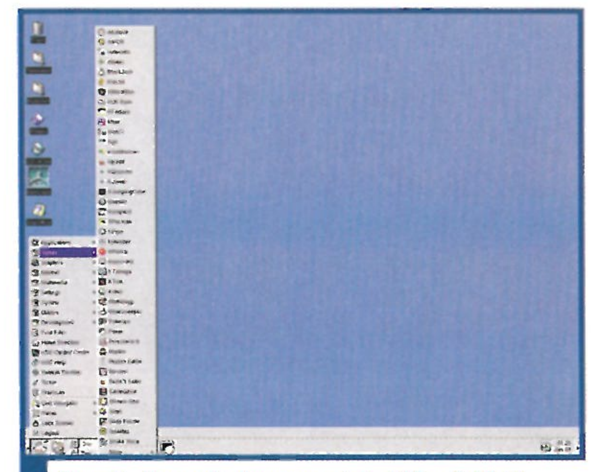
SUSE 6.3

De origen alemán, *SuSE 6.3* es la última versión de esta distribución que ha aparecido en el mercado. Esta distribución está producida por la empresa *SuSE Gmbh* contando con un enfoque muy centrado en el usuario final y con aspiraciones de ser un sistema operativo lo más completo y sencillo de manejar como sea posible. Una de las curiosidades que rodean a esta distribución está en que, en ocasiones, sus ventas en el mercado alemán se pueden equiparar o superar a las de *Windows 98* (igual que le ocurre a *TurboLinux* en Japón).

SuSE se encuentra disponible en varias versiones: una de libre distribución muy recortada (ocupa un único *CD-ROM*) y otra comercial muy completa. Esta última es la más difundida de las dos y por ello es la que vamos a usar como referencia a lo largo de este análisis. Actualmente, el soporte multiplataforma de *SuSE* es muy pobre ya que sólo podemos encontrar una versión beta de *SuSE 6.1* para máquinas *Alpha*.

Aunque pensada para trabajar con Gnome, RedHat incluye también KDE 1.1.2

SuSE 6.3 se presenta en una caja acompañada por un manual de 550 páginas, 6 *CD-ROMs* que contienen la distribución y dos disquetes de arranque para aquellos usuarios que no disponen de soporte para *CD-ROM* en sus máquinas. Adicionalmente se puede comprar esta misma distribución pero en formato *DVD-ROM*, lo que elimina la necesidad de cambiar de discos mientras se instala.


Figura 9.- Mostrando algunos menús de *KDE* en *SuSE* 6,3.

El manual está correctamente impreso, y lo más importante, está completamente traducido al castellano (eso sí, como siempre hay algunas incorrecciones a lo largo del mismo) y cubre la instalación, la configuración y los primeros pasos con el sistema operativo todo ello explicado con un lenguaje adecuado, utilizando sólo los tecnicismos justos y tratando de conseguir una explicación clara y concisa.

SuSE ofrece un sistema básico de prestaciones similares a *RedHat*, utilizando las librerías *glibC 2.1* y un *kernel 2.2.13*, ligeramente más moderno que el de *RedHat*, así como el compilador *EGCS* en sustitución del conocido *gcc*.

Uniéndose a la actual tendencia de incluir instaladores gráficos, muy sencillos de manejar y aptos para usuarios inexpertos, *SuSE* se instala mediante su propio instalador gráfico, llamado *YaST 2*. Éste es muy sencillo de manejar y permite seleccionar el idioma en el que vamos a realizar la instalación.

SuSE incorpora el kernel 2.2.13 y las librerías glibc 2.1, así como un instalador gráfico muy sencillo

El proceso de instalación es, como ya se ha mencionado anteriormente, extremadamente sencillo al contar en todo momento con una ayuda que explica exactamente lo que representa cada una de las opciones de instalación y los parámetros que le damos al sistema. En algunos casos el procedimiento puede ser aún más sencillo, ya que podemos elegir una instalación automática que ahorra algunos pasos del proceso de instalación.

Durante el proceso de instalación sólo hemos encontrado un problema, y es que el espacio en el que se ha de instalar el sistema debe ser contiguo y estar situado al final de las particiones existentes, en caso de no cumplirse esta premisa, el programa nos dirá que no tenemos espacio suficiente para instalar el sistema. Este problema está documentado en el manual y actualmente están trabajando para que la próxima versión del instalador no sufra este inconveniente.

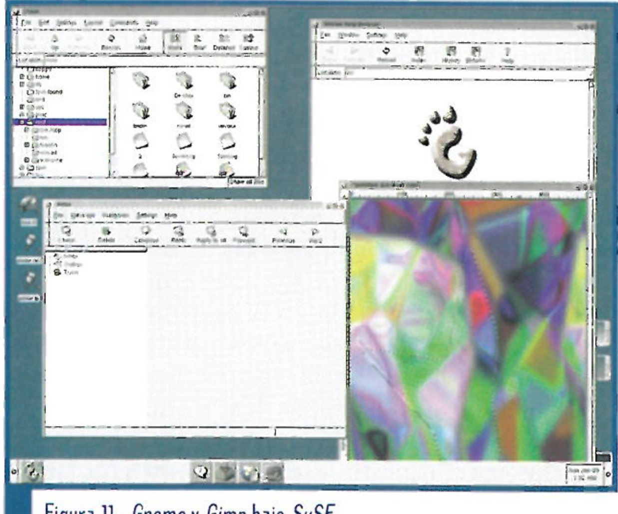
SuSE cuenta como una de sus referencias más importantes el hecho de haber batido en ventas a Windows 98 en Alemania

Durante el proceso de instalación propiamente dicho el sistema avisa de cuándo hemos de cambiar de *CD-ROM*, y aunque el proceso resulta un poco largo, no es complicado de seguir.

Una vez finalizada la instalación el programa configura automáticamente los parámetros más habituales del sistema, dejando los más específicos (como configuración de la conexión a red y similares) para más tarde, cuando ya tengamos el sistema básico funcionando a nuestro gusto.

SuSE ofrece posiblemente el más amplio surtido de gestores de ventanas y escritorios de todas las distribuciones, incluyendo *KDE* y *Gnome* como escritorios y toda una legión de gestores de ventanas entre los que elegir, incluyendo desde los más actuales y conocidos (*enlightment*, *kwm*) hasta algunos de muy poca difusión (*9wm*). Con esta amplia capacidad de elección no sólo podemos elegir el aspecto de nuestro entorno gráfico, sino que podemos usar gestores de ventanas muy ligeros para cuando trabajemos en máquinas con pocos recursos.

La distribución está muy bien surtida en cuanto a herramientas y material para desarrolladores, posiblemente mejor que *Debian*. Entre las aplicaciones incluidas se encuentra *Kdevelop 1.0B3*, edición personal de *Adabas 11*, versión de demostración de *VMWARE*, numerosas librerías, así como el código fuente de todos los paquetes de libre distribución incluidos en *SuSE*. En el campo de los compiladores se incluyen la habitual *suite* de compiladores *GNU*, intérpretes de *Perl* y *Python*, así como versiones de demostración de algunos productos como *NetBeans* o el compilador *Flagship* (para la adaptación de programas escritos en *Clipper*). A toda esta legión de programas y productos debemos añadirle que, al igual que *RedHat*, *SuSE* ha conseguido que se realicen versiones


Figura 11.- *Gnome* y *Gimp* bajo *SuSE*.

adaptadas de algunos productos de desarrollo profesionales como *CodeWarrior*, *Adabas 11*, etc.

SuSE también ha conseguido que diversos fabricantes creen aplicaciones adaptadas a su distribución

Esta unión de factores convierten a *SuSE 6.3* en un producto altamente recomendable para desarrolladores y para usuarios finales. Los desarrolladores reciben un producto fiable, sencillo de manejar y que incluye gran cantidad de herramientas para la programación con un funcionamiento y estabilidad impecables. Los usuarios finales reciben un sistema sencillo de instalar y administrar, con un manual comprensible y un servicio de soporte excelente.

COMPARACIÓN DE DISTRIBUCIONES

Las diferencias entre las diversas distribuciones de cara al desarrollador son mínimas, ya que a excepción del *software* comercial, podemos conseguir que cualquier programa funcione adecuadamente en cualquier distribución simplemente añadiendo las librerías necesarias o los compiladores adecuados.

El *software* comercial también funcionará añadiendo las librerías adecuadas en la mayoría de los casos, pero siempre existe la posibilidad de que no funcione correctamente, y tendremos que contar, en ese caso, con que el servicio de soporte del fabricante generalmente no nos ayudará a solucionar el problema, al tratarse de una plataforma no soportada.

La dificultad de la instalación es un factor más determinante, ya que puede influir en el coste final de la implantación de una solución comercial desarrollada en *Linux*. Esto está íntimamente relacionado con la dificultad de manejo del entorno y la curva de aprendizaje del mismo, factores que también repercuten en el coste, además de limitar el perfil de los usuarios.

Los factores que más nos van a influir no son los programas incluidos, sino los que influyen en el despliegue de la aplicación creada

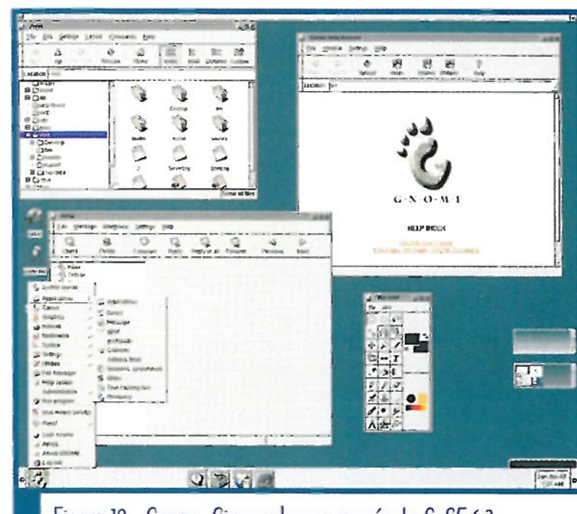
Adicionalmente también interesa considerar factores como la disponibilidad en múltiples plataformas *hardware*, estabilidad del sistema, disponibilidad de soporte comercial de la distribución y nuestra propia comodidad en el trabajo diario con dicha distribución.

Teniendo en cuenta estos factores y las características de las distribuciones comentadas anteriormente podríamos recomendar *Debian* como plataforma de desarrollo para aplicaciones de *software* gratuito, en múltiples plataformas *hardware* o muy críticas, que requieran una estabilidad a prueba de bomba.

En la misma tónica *RedHat* y *SuSE* quedarían más enfocadas al desarrollo de aplicaciones con herramientas comerciales, o dirigidas al usuario final, ya que tradicionalmente no son consideradas tan estables como *Debian* y no están disponibles en la cantidad de plataformas en las que se puede encontrar a *Debian*.

CONCLUSIÓN

La conclusión de este análisis sería que utilizar una distribución u otra no representa un cambio importante desde el punto de vista del desarrollo de las aplicaciones, ya que las *API* del sistema no van a cambiar, sino que van a ser factores relacionados con el despliegue de la aplicación los que van a determinar la elección de la distribución, pero nunca de forma determinante y tajante. Esta interoperabilidad entre diversas distribuciones es una de las grandes ventajas de *Linux* y uno de los motivos por los que su avance y su desarrollo tecnológico es tan rápido.


Figura 12.- *Gnome*, *Gimp* y algunos menús de *SuSE* 6.3.





Visual C++ y MFC (I)

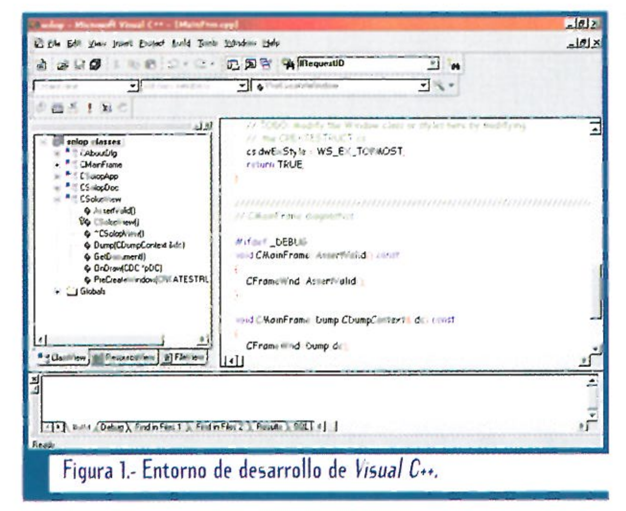
Constantino Sánchez Ballesteros.
Técnico Superior Desarrollo Aplicaciones Informáticas.

Comenzamos una nueva sección dedicada a la programación de las *MFC* bajo *Visual C++*. Con estas librerías de *Microsoft* seremos capaces de realizar aplicaciones profesionales con el menor esfuerzo.

INTRODUCCIÓN

Microsoft Visual C++ es un entorno de programación para *Windows* que utiliza la combinación de la programación orientada a objetos y el *SDK* de *Windows* para la creación de aplicaciones gráficas.

Las aplicaciones *Windows* son sencillas de utilizar, pero puedo certificar que el desarrollo de las mismas no es tan sencillo. Debido a este último punto, *Visual C++* incorpora una serie de herramientas que nos permitirán realizar dichas aplicaciones de una forma más fácil e intuitiva.


Figura 1.- Entorno de desarrollo de *Visual C++*.

El mayor avance en este sentido que ha implementado *Microsoft* son las ya famosas *MFC (Microsoft Foundation Classes)*. Estas clases permitirán crear potentes aplicaciones *Windows* y manejar sus componentes como objetos. Recordemos que un objeto tiene una serie de propiedades, métodos y funciones, que además podremos reutilizar para crear nuevos objetos que heredan propiedades de los objetos referenciados.

Por ejemplo, para crear una caja de diálogo tenemos a nuestra disposición la clase *CDialog*. Cuando queramos construir nuevas cajas de diálogo para nuestra aplicación sólo tendremos que crear un nuevo objeto que referencie a esa clase y tendremos a nuestra disposición todos los atributos de la clase principal en nuestra nueva caja de diálogo.

Visual C++, cuando se utiliza para desarrollar aplicaciones para *Windows*, se centra en dos tipos de objetos:

- Ventanas
- Controles (visuales y no visuales)

Utilizando estos objetos podemos diseñar sin escribir nada de código, una *interfaz* gráfica para una aplicación. El problema viene cuando queramos dar funcionalidad a esa *interfaz*, que requerirá algunas líneas de código.

Esto es, para realizar una aplicación se crean ventanas, a veces llamadas formularios, y sobre ellas

se dibujan otros objetos llamados controles. A continuación se escribe el código fuente relacionado con la función que tiene que realizar cada objeto. Esto quiere decir que cada objeto (ventanas y controles) está ligado a un código determinado que no se ejecuta hasta que se dé el evento que lo active. Por ejemplo, podemos programar un botón (objeto que se puede pulsar) para que responda a una pulsación del ratón sobre él, ejecutándose en ese momento una acción determinada que tengamos definida en el programa.

A continuación se detallan, a modo de resumen, las características principales que incorpora *Visual C++:*

- Una biblioteca de clases, *MFC*, que incluye soporte para ventanas, cajas de diálogo, controles como etiquetas, cajas de texto, botones de pulsación y objetos *GDI (Interfaz* gráfica de *Windows)* como lápices, pinceles, fuentes de letras y mapas de *bits (bitmaps).* Recordemos que los objetos se comunican entre sí mediante mensajes, los cuáles también están soportados por las *MFC*.
- Asistentes para el desarrollo de aplicaciones como *AppWizard,* editores de recursos (editor de menús, de diálogos, de tablas de cadenas de caracteres, de tablas de aceleradores y de objetos gráficos como mapas de *bits* o iconos).
- Intercambio de datos con otras aplicaciones *(DDE - Dynamic Data Exchange).*
- Creación y utilización de bibliotecas dinámicas *(DLL - Dynamic Link Libraries).*
- Galería de objetos incrustados y vinculados (*OLE - Object Linking and Embedding).*
- Visualización y manipulación de datos de otras aplicaciones *Windows* utilizando controles *OLE*.
- Una *interfaz* para múltiples documentos que permite crear una aplicación con una ventana de aplicación y múltiples ventanas de documento, como por ejemplo, *Microsoft Word (MDI - Múltiple Document Interface).*
- Objetos de acceso a datos (*DAO*) que permiten acceder a bases de datos a través del motor de *Access o* de controladores *ODBC*.
- Cabeceras precompiladas que reducen el tiempo de compilación.
- Un entorno de desarrollo integrado (editor de texto, compilador, depurador, explorador de código fuente, administrador de proyectos, etc).
- Soporte para el estándar *COM (Component Object Model)* al que pertenecen los componentes activos *(ActiveX* –anteriormente denominados controles *OLE*-).
- Soporte para *Internet.*

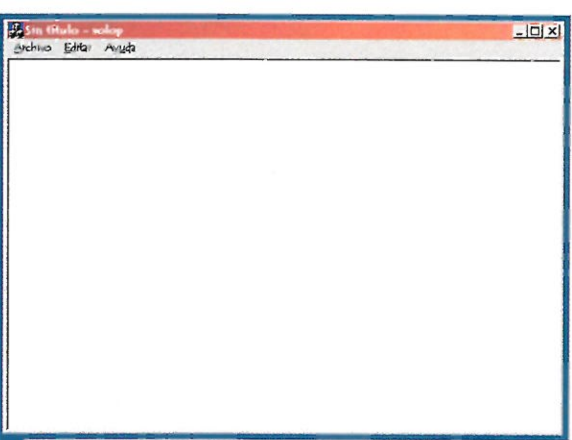


Figura 2.- Programa de ejemplo desarrollado en este artículo.

NUEVAS CARACTERÍSTICAS EN LA VERSIÓN 6.0

Como es lógico, en cada nueva versión se implementan nuevas opciones y funcionalidades. En las siguientes líneas detallaremos algunas de las más interesantes.

DEBUGGER

- **Editar y continuar:** permite incorporar ediciones durante una sesión de depurado del programa sin tener que salir de la sesión, reconstruyendo y reiniciando el depurador. Los cambios son recompilados y aplicados a la aplicación que se ejecute en memoria.
- **Visualización de registros MMX:** ahora es posible visualizar registros *MMX* en las ventanas *Watch* y *Quickwatch* utilizando los símbolos *MM0-MM7*. Los registros *MMX* son enteros de 64 *bits* y se visualizarán sobre todas las máquinas *x86,* siempre que éstas soporten el juego de instrucciones de *Intel*.
- **GUIDs decodificados para su visualización:** estas claves en forma de números llamados *GUIDS* (incluyendo *IIDs, CLSIDs,* y *REFIIDs)* pueden visualizarse por nombre (si se encuentra en el registro), o por defecto, mostrando el detalle hexadecimal del mismo.

EDITOR

- **Soporte de controles Internet Explorer 4.0 en el editor de recursos:** es posi-

ble utilizar este soporte con las *MFC* o sin ellas. Para acceder a las clases de los controles de *IE 4.0* lo haremos vía "barra de herramientas *Controls*" del editor de diálogos, agregando estos controles a nuestros formularios y cajas de diálogo.

PROYECTO

- **Comando New Form:** este comando es nuevo en *Visual C++ 6.0*. Genera un formulario a partir del menú *Insert*. Este comando es esencialmente un asistente de clases para *CFormView, CDialog,* y las clases vista de bases de datos. Al igual que *ClassWizard*, el comando *New Form* también crea una caja de diálogo con todas las opciones apropiadas.

VENTANAS Y CONTROLES

Una forma sencilla de crear una ventana es utilizando el editor de diálogos de *Visual C++*. Otra forma menos sencilla es escribiendo directamente el código que dé lugar a la misma. Una ventana sirve de fondo para los controles y para los gráficos. Se pueden utilizar tantas ventanas como se necesiten y dependiendo de la utilidad que se las dé, éstas pueden ser de diferentes tipos. Por ejemplo, ventana principal, ventana de diálogo, ventana de presentación, vista del documento etc.

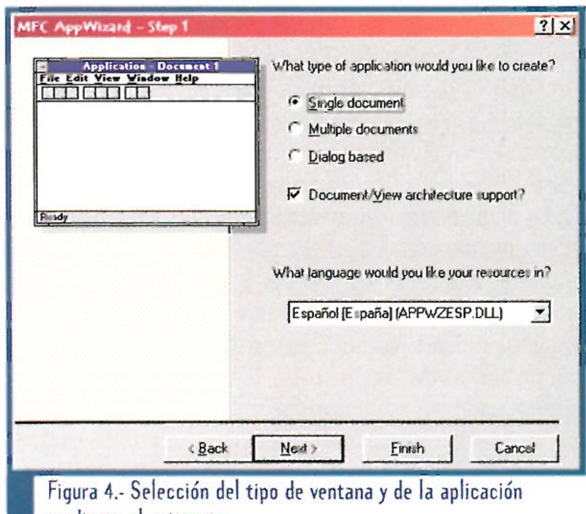


Figura 4.- Selección del tipo de ventana y de la aplicación mediante el asistente.

Los controles son objetos que podemos dibujar sobre la ventana, tales como etiquetas, cajas de texto, casillas de verificación, botones de opción, botones de pulsación, etc, y su finalidad es la de aceptar datos o visualizarlos.

Los asistentes como AppWizard generan aplicaciones sin escribir una línea de código

Para dibujar los controles se utiliza la caja de herramientas del editor de diálogos de *Visual C++*. También es posible incluir cualquier imagen existente o diseñada a medida con el editor de objetos gráficos, como por ejemplo un icono, un mapa de *bits* o un fichero con formato *.WMF, .GIF* o *.JPEG*. Otra forma más complicada y menos intuitiva de dibujar los controles sería escribiendo código para crear cada uno de ellos durante la ejecución.

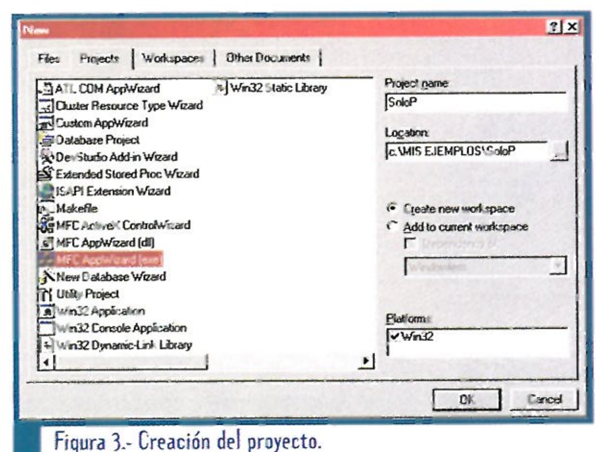


Figura 3.- Creación del proyecto.

Una ventana puede también incluir menús desplegables y flotantes, para proporcionar al usuario un amplio rango de órdenes seleccionables. *C++* junto con la biblioteca *MFC* permite crear aplicaciones de muy diversos tipos. Por ejemplo, bases de datos, control de procesos, representaciones gráficas, aplicaciones multimedia, aplicaciones para *Internet*, etc.

ENTORNO DE DESARROLLO DE VISUAL C++

Microsoft Visual C++ es un entorno integrado de desarrollo para aplicaciones C y C++ sobre diversas plataformas. *Visual C++* ofrece un entorno de desarrollo de *C/C++, Developer Studio,* integrado en *Windows*, que permite construir aplicaciones *Windows (.EXE),* aplicaciones de consola *(.EXE),* librerías de enlace dinámico *(.DLL),* librerías estáticas *(.LIB),* modelos *AppWizard* personalizados *(.AWX),* controles activos *(ActiveX)* y otros.

En la Figura 1 podemos ver el entorno de desarrollo.



VENTANA DE PROYECTO

La ventana *Workspace* (ventana de la izquierda de la Figura 1) permite una cuidada gestión de las componentes de una aplicación. Basta con seleccionar un miembro de una clase, un recurso o un fichero independiente de la aplicación y *Developer Studio* se encarga del resto.

Una clase podemos tomarla como un objeto que alberga funciones, procedimientos...

Cuando se crea o se abre una aplicación o proyecto *(Workspace)*, *Developer Studio* visualiza los componentes de ese proyecto en la ventana *Workspace*. En la parte inferior de dicha ventana se visualizan cuatro pestañas que dan acceso a otras tantas vistas. Cada vista tiene, al menos, una carpeta que contiene los elementos a los que hace referencia la pestaña. Estas vistas son:

- **ClassView:** visualiza las clases que componen la aplicación. Haciendo doble clic en el nombre de una clase se visualiza la declaración de la misma en la ventana de edición. Haciendo clic en el símbolo + se visualizan los miembros de esa clase y si hacemos el doble clic en un miembro, se visualiza la definición (código) del mismo en la ventana de edición.
- **ResourceView:** visualiza los recursos que forman parte de la aplicación. La forma de acceder a los recursos es similar a la explicada en el punto anterior, excepto que ahora, el editor de código es sustituido por el editor del recurso sobre el que se ha hecho doble clic (acelerador, diálogo, icono/*bitmap*, menú, tabla de cadenas de caracteres, barras de herramientas o versión).
- **FileView:** visualiza los ficheros fuente que componen la aplicación. Haciendo doble clic en el nombre de cualquiera de ellos, se visualiza la ventana de código correspondiente al mismo.

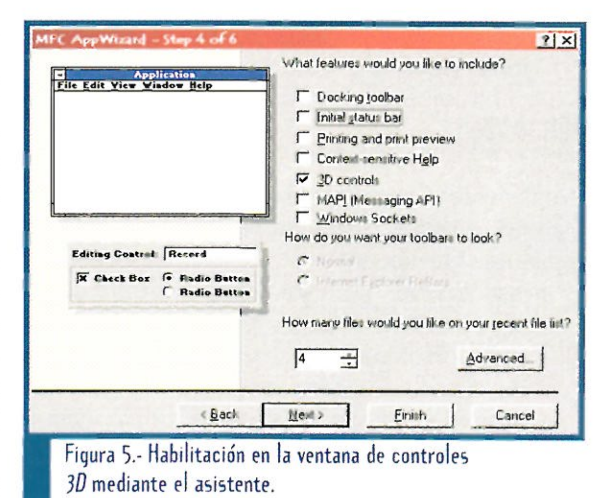


Figura 5.- Habilitación en la ventana de controles *3D* mediante el asistente.

VENTANA DE EDICIÓN

A la derecha de la ventana *Workspace*, en la Figura 1, está la ventana de edición. Puede haber varias ventanas de edición abiertas simultáneamente. Por ejemplo, para visualizar el código de diferentes ficheros, visualizar diferentes recursos o visualizar temas de ayuda en línea. Para cambiar de una ventana de edición a otra podemos utilizar las teclas **Ctrl+F6**.

En nuestra aplicación podremos incluir cualquier imagen o diseñarla

Justamente encima de la vista de la ventana de edición se visualiza una barra de herramientas con tres listas desplegables y dos botones de pulsación, denominada *WizardBar*. Su función es proporcionar una forma fácil de añadir nuevos manipuladores de eventos o saltar a algún manipulador existente, tareas propias de *ClassWizard*. El resto de barras de herramientas de *Microsoft Developer Studio* las podemos ver/ocultar ejecutando la orden *Customize...* del menú *Tools* y haciendo clic en la pestaña *Toolbars*.

VENTANA DE SALIDA

Por último, debajo de las ventanas *Workspace* y de edición, aparece otra ventana de salida de resultados, denominada *Output window*. Algunas de sus vistas son:

- **Build:** visualiza los resultados del proceso de compilación y enlace de una aplicación.
- **Debug:** visualiza información resultante de un proceso de depuración. Por ejemplo, mensajes de afirmación *(assertion)*, mensajes producidos por el objeto *afxDump*. etc.
- **Find in Files:** visualiza los resultados de la búsqueda de un determinado texto en varios ficheros, realizada por medio de la orden *Find in Files* del menú *File* de *Developer Studio*.
- **Profile:** visualiza el perfil de un programa después de su ejecución. En este perfil podemos ver qué funciones se ejecutan, cuánto tiempo se ha empleado en cada una, cuántas veces se ejecutan, etc. Para disponer de esta información es necesario habilitar la opción *Enable profiling* del enlazador (orden

Settings del menú *Build,* pestaña *Link).*

UTILIDADES

También desde *Developer Studio* se puede acceder fácilmente a las siguientes herramientas:

- **AppWizard:** esta herramienta, accesible a través de la opción *Workspace* que se visualiza cuando se ejecuta la orden *New* del menú *File* de *Developer Studio,* permite crear el esqueleto básico de una aplicación a partir de la biblioteca de clases *MFC* incluida en *Visual C++.* Este esqueleto está compuesto por una serie de ficheros fuente y ficheros de recursos. Los ficheros fuente contienen el código *C++* que forman la aplicación y los ficheros de recursos contienen la descripción de los recursos que utiliza la aplicación.
- **Editores de recursos:** estas herramientas son accesibles desde la vista *ResourceView* de la ventana *Workspace.* Permiten editar la tabla de aceleradores, las ventanas de diálogo, los iconos, los menús, la tabla de cadenas de caracteres, las barras de herramientas o la versión de la aplicación. Una característica importante de los editores de recursos es su integración con *ClassWizard* para unir el código a los objetos de la *interfaz* gráfica del usuario, con el fin de dotarles de la funcionalidad correspondiente.
- **ClassWizard:** esta herramienta, accesible desde el menú *View* de *Developer Studio,* opcionalmente desde la barra de herramientas (las barras de herramientas se pueden personalizar a través de la orden *Customize* del menú *Tools),* o pulsando las teclas **Ctrl+W,** permite generar automáticamente las clases y funciones que gestionan los mensajes de los objetos *Windows.* Se utiliza para conectar los controles de la aplicación con el código que se ha de ejecutar al activar dichos controles.

Otras herramientas de soporte para la programación también incluidas son:

- **Depurador:** como su nombre indica, permite depurar una aplicación ejecutándola paso a paso. Para acceder a esta herramienta, hay que seleccionar el submenú *Start Debug* del menú *Build* de *Developer Studio* o utilizar barra de herramientas *Debug.* Las barras de herramientas se pueden hacer visibles u ocultar desde la orden *Toolbars* del menú *View.*
- **SPY ++:** es una herramienta desarrollada para *Windows* que permite visualizar las características de una ventana (estilo, dimensiones, ventana padre, etc.) y los mensajes que recibe esa ventana. Se ejecuta desde el menú *Tools.*
- **MFC Tracer:** esta herramienta se arranca desde el menú *Tools* y permite activar o desactivar las opciones que determinan la categoría de los mensajes enviados desde la versión de depuración de una aplicación *MFC* a la ventana de depuración.

Figura 6.- Diseño del menú.

NUESTRA PRIMERA APLICACIÓN

En este apartado, nuestro objetivo será crear nuestra primera aplicación con *Visual C++.* El objetivo es presentar en pantalla un formulario como el de la Figura 2 pero sin escribir nada de código. Esto demostrará la facilidad de uso que proporcionan las *MFC* para la creación de programas en *C++* bajo *Windows.*

Por supuesto, más adelante iremos añadiendo código al programa para dotarle de mayor funcionalidad.



Cuando nos proponemos hacer una aplicación, lo primero que debemos tener claro es la interfaz de usuario que vamos a utilizar. Si no vamos a implementar menús ni utilizar documentos la elección será utilizar una ventana tipo diálogo. En caso contrario, utilizaremos un formulario con soporte *Documento/Vista.* También será posible crear programas tipo consola, *DLL's,* etc.

El propio programa en C++ crea aplicaciones sin haberlas especificado

Para nuestro primer ejemplo, vamos a crear una aplicación *Documento/Vista* que incluya un menú para salir de la aplicación y visualizar una caja de diálogo mostrándonos el *Copyright* del programa. Reitero que todo esto se hará sin una línea de programación.

Para comenzar a realizar la aplicación seguiremos los siguientes pasos:

- Seleccionamos la opción **New** del menú **File.** Aparecerá en pantalla una ventana con diversas opciones (Figura 3). Puesto que nosotros vamos a comenzar un nuevo proyecto pulsaremos sobre la pestaña *Projects*. Dentro de esta pestaña se mostrarán diversos tipos de aplicaciones, librerías, etc, que podremos crear. En nuestro caso, seleccionaremos **MFC AppWizard (EXE).** Debemos dar un nombre al proyecto (en el ejemplo se utilizó el nombre **Solop**) y establecer el directorio por defecto del mismo.

- Ahora es el momento de establecer las propiedades que tendrá el formulario (Figura 4). En este caso utilizaremos un documento (equivalente a una ventana), activaremos la opción *Document/View architecture support* para el soporte documento/vista y seleccionaremos el lenguaje que vamos a utilizar (español).

- Pulsamos sobre el botón **Next** (siguiente) y dejamos los siguientes campos tal y como vienen por defecto hasta que lleguemos al paso 4 del asistente (Figura 5). En él sólo habilitaremos **3D controls** y pulsaremos sobre el botón **Finish** para acabar el desarrollo inicial de la aplicación.

- Una vez llegado a este punto tendremos la aplicación preparada para compilar y ejecutarse. Para poder ver los resultados en pantalla, en primer lugar tendremos que compilar el programa. Esta acción la efectuaremos mediante la pulsación simultánea de las teclas **Ctrl+F5** o seleccionando la opción **Execute Solop.exe** del menú **Build. Solop.exe** será el nombre que utilicemos para nombrar al ejecutable. Este nombre es tomado automáticamente al establecer el nombre al proyecto.

Siguiendo estos pasos hemos visto cómo desarrollar una simple pero funcional aplicación en *C++* sin escribir nada de código. Una vez ejecutado el programa, si pulsamos la opción **Acerca de Solop...** del menú **Ayuda**, aparecerá automáticamente una caja de diálogo con el botón **Aceptar** que visualiza el *copyright* del programa y su nombre. Esto también lo ha creado de forma automática el asistente, y como veremos más adelante, será fácil modificarlo a nuestro gusto.

Lo que vamos a hacer a continuación será eliminar las opciones del menú que no interesen. Para este caso en particular, sólo dejaremos las opciones **Ayuda** y **Archivo.** Esta última la cambiaremos por el nombre **Aplicación.** Lo que se pretende conseguir es que cuando se pulse sobre la nueva opción **Aplicación** se desglose un submenú con la opción **Salir**, para poder salir de forma automática del programa.

Manos a la obra. Lo primero que tendremos que hacer será abrir el editor de recursos. Esto se hace pulsando sobre la pestaña *Resource View* de la ventana *Workspace*. Aparecerá la carpeta *Solop resources*, que abriremos, de tal forma que aparecerán otras tantas carpetas visualizando los diferentes recursos que tenemos asignados al programa. Para esta ocasión, abriremos la carpeta menú y su contenido, tal y como muestra la Figura 6.

Primero debemos eliminar la opción Editar del menú incluyendo todo su contenido. Para ello, pulsaremos con el ratón sobre dicha opción y pulsaremos la tecla **Supr** (borrar)

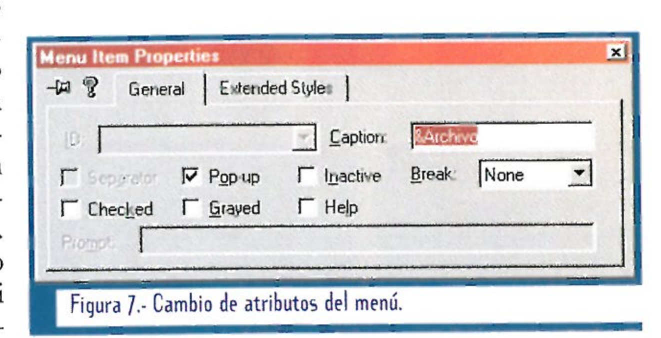
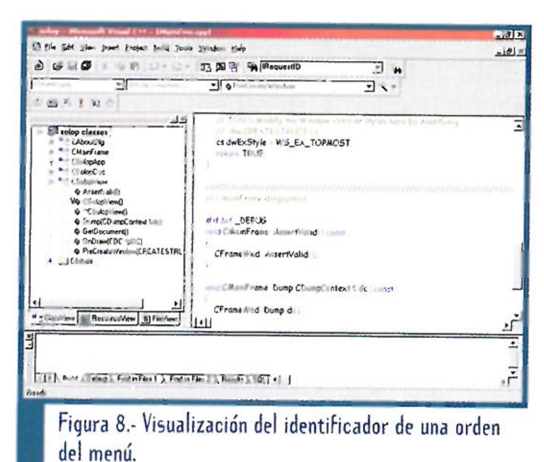


Figura 7.- Cambio de atributos del menú.

SÓLO PROGRAMADORES


Figura 8.- Visualización del identificador de una orden del menú.

para eliminar esta clave del menú. Para el menú **Archivo** pulsaremos sobre él y eliminaremos todas las claves que no deseemos. En nuestro caso, sólo deben quedar la clave principal **Archivo** y la subclave **Salir**.

Puesto que hemos determinado cambiar el atributo **Archivo** por el de **Aplicación**, pulsaremos con el botón derecho sobre la clave **Archivo** del menú para obtener sus propiedades. Aparecerá una pantalla como la de la Figura 7. En ella, sustituiremos el campo *Caption* que contiene el dato **&Archivo** por el de **&Aplicación**. El símbolo **&** (*ampersand*) se utiliza para subrayar la primera letra que encuentre después de este carácter y se tenga acceso rápido mediante teclado (combinación de teclas) a esta opción del menú.

La opción **Salir** de la clave del menú **Aplicación** permitirá finalizar de forma automática la aplicación. Para ello, utiliza una constante o identificador *ID (ID_APP_EXIT)* que permite este proceso. Esto puede verse en las propiedades de su clave. Por supuesto, si cambiamos esta constante por otra de las que tiene establecidas el sistema el resultado de la pulsación de esta opción del menú será muy diferente (véase Figura 8).

CAMBIO DE LA CAJA DE DIÁLOGO

Para poder cambiar el contenido de la caja de diálogo que *Visual C++* ha establecido por defecto, también tendremos que utilizar el editor de recursos. La única variación es que tendremos que abrir la carpeta *Dialog*. Haciendo doble clic sobre *IDD_ABOUTBOX* aparecerá en el editor la ventana correspondiente (Figura 9). Mediante los controles situados dentro de la barra de herramientas correspondiente podemos insertar dentro del diálogo el que queramos.

Una clase podemos tomarla como un objeto que alberga funciones, procedimientos...

En este caso se ha optado por incluir una etiqueta, mediante la que se añadirá el nombre del autor de la aplicación a la ventana para su posterior visualización. Ni que decir tiene que podemos cambiar el texto que está establecido en los controles asignados previamente al formulario.

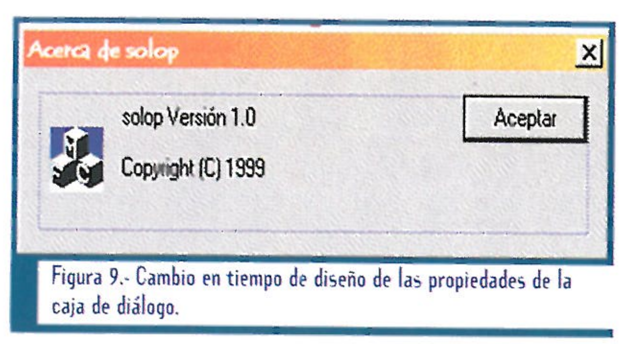


Figura 9.- Cambio en tiempo de diseño de las propiedades de la caja de diálogo.

CAMBIO DEL NOMBRE DE LA APLICACIÓN

Para poder cambiar el título que tiene por defecto la aplicación debemos hacer lo siguiente:

- Abrir el editor de recursos (anteriormente utilizado).
- Hacer doble clic en el recurso **String Table** o pulsar sobre el botón **abc** de la barra de herramientas.
- Visualizaremos las propiedades de la cadena *IDR_MAINFRAME* pulsando con el botón derecho sobre ese identificador y utilizando la opción **Propierties**.
- En esta cadena encontraremos siete subcadenas separadas por **/n**. La primera corresponde al título de la aplicación. Por lo tanto, cambiando el contenido de ésta, cambiaremos automáticamente el título de nuestro programa.

Con el editor de recursos podremos cambiar el contenido de la caja de diálogo

Aunque en nuestra aplicación no utilicemos el documento, éste existe, de ahí que la barra de título de la aplicación se visualice como nombre del mismo **Sin título**. Para que este título no se visualice, utilizaremos un espacio en blanco como segunda cadena del recurso *IDR_MAINFRAME* de *string table*.

Hasta aquí llega nuestra primera aplicación, en las próximas entregas continuaremos descubriendo muchas de las posibilidades que nos ofrece este apasionante entorno de programación.



Aplicaciones Web con acceso a BBDD basadas en Java (II)

Adolfo Aladro García.
Analista Programador

En el capítulo anterior se analizaron las posibles arquitecturas de las aplicaciones cliente-servidor basadas en *servlets* y *JDBC*. Ahora se estudiarán los *servlets* en mayor profundidad para conocer a fondo la naturaleza de éstos y su integración con *JDBC*.

CICLO DE VIDA DE UN SERVLET

Los *servlets*, al igual que los *applets*, tienen un ciclo de vida predeterminado. Su conocimiento es algo indispensable si queremos sacar provecho a toda la potencia y versatilidad que proporcionan. A grandes rasgos podríamos hablar de tres fases: inicialización, proceso de peticiones y la fase de terminación.

- **Inicialización**

Un *servlet* puede ser inicializado de tres formas posibles:

- Automáticamente cuando el servidor arranca por primera vez, si así se ha configurado. Esta opción depende por completo de las características propias del servidor encargado de dar soporte a los *servlets*.
- La primera vez que un cliente realiza una petición. Éste suele ser el caso más frecuente.
- Cuando el *servlet* se vuelve a cargar. Esto puede suceder, por ejemplo, cuando éste ha sido modificado y compilado de nuevo. Si esto ocurre, cuando llega una nueva petición el servidor, éste último se da cuenta de que el *servlet* ha sido recompilado y entonces lo carga de nuevo en memoria.

Una vez que el *servlet* ha sido cargado, el servidor crea una instancia del mismo y llama al método



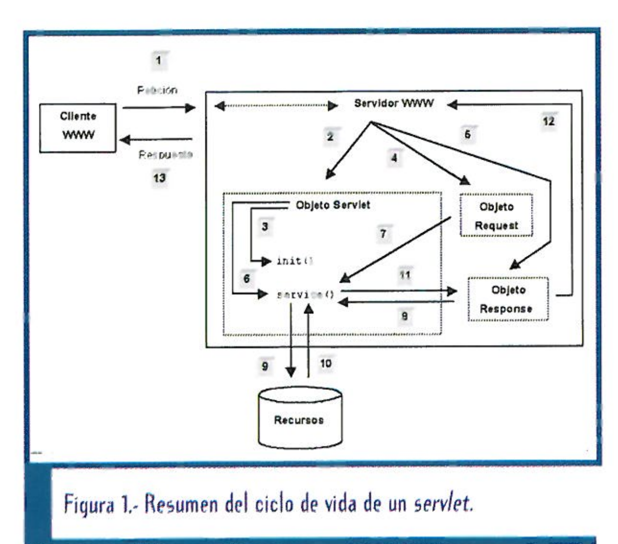


Figura 1.- Resumen del ciclo de vida de un *servlet*.

init. Este método será invocado una sola vez con independencia del número de peticiones que lleguen al *servlet*. Esto significa que todas las tareas que tengan un propósito global dentro de la aplicación *Web* a la que pertenece ese *servlet* deben ser llevadas a cabo dentro de este método: configuración de parámetros globales, inicialización de variables estáticas compartidas por todos los *threads* del *servlet*, etc.

Los servlets se ejecutan por fases

- **Proceso de las peticiones**

Por cada petición, el servidor crea un objeto *Request* (petición) y un objeto *Response* (respuesta). En ese momento llama al método *service* del *servlet*, pasándole como parámetros ambos objetos. El método *service* toma los datos de la petición del objeto *request* (éstos pueden ser por ejemplo los datos procedentes de un formulario o la información que transporta la propia dirección *URL*), procesa la petición y utiliza los métodos del objeto *Reponse* para enviar la información de vuelta al cliente. Lo más usual es que ésta sea una página *HTML* aunque también se puede responder con cualquier tipo de datos.

El método *service* responde a una abstracción que ofrece la clase *GenericServlet*. Con frecuencia suele ser sustituido por los métodos *doGet* o *doPost*, que están directamente relacionados con el protocolo *HTTP*, aunque su funcionamiento es exactamente el mismo. Estos métodos pertenecen a la clase *HttpServlet* que depende directamente de la clase *GenericServlet*.

El método service es una abstracción

Es importante incidir en que las distintas peticiones se atienden de forma concurrente. Esto permite la coordinación de actividades entre varios clientes. Es posible incluso utilizar variables estáticas para mantener una zona común de información a todos los clientes, tal y como se empezó a estudiar en el primer artículo de esta serie.

- **Terminación**

Cuando el servidor no necesita el *servlet*, o el *servlet* va a ser cargado de nuevo, el servidor llama al método *destroy*. Este método se encarga de liberar todos los recursos que el *servlet* consume. Como en los casos anteriores, este método puede ser sobreescrito para realizar las operaciones que se consideren necesarias antes de que el *servlet* muera".

La Figura 1 muestra un resumen del ciclo de vida de un *servlet*:

- El cliente realiza una petición (1). Lo más usual es que la carga del *servlet* se haga en este momento, como consecuencia de esa primera petición por parte de un cliente, aunque también existe la posibilidad de configurar el servidor de forma que por defecto se carguen algunos *servlets* cuando éste sea arrancado.
- El servidor crea una instancia del *servlet*, o lo que es lo mismo, un objeto *servlet* (2).
- El servidor llama al método *init* del *servlet* (3).

Los tres puntos anteriores solamente suceden una vez al iniciarse el *servlet*. A partir de ese momento todas las peticiones son atendidas por el mismo objeto *servlet*. El servidor crea para cada una de ellas un *thread* o hilo de ejecución distinto. Así, cada vez que el servidor reciba una petición ocurre lo siguiente:

- El servidor crea un objeto *Request* que almacena la información relativa a la petición (4).
- El servidor crea un objeto *Response* que hace posible la respuesta hacia el cliente por parte del servidor (5).

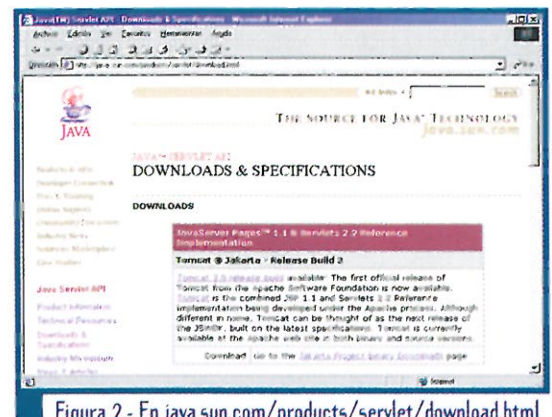


Figura 2.- En java.sun.com/products/servlet/download.html podemos descargar la documentación relativa a la *API* de Java servlets.

- El servidor llama al método *Service* (o *doGet*, o *doPost*) del *servlet* (6), y le pasa como parámetros los objetos *Request* (7) y *Response* (8).
- El método *Service* procesa la petición accediendo a cualquier tipo de recurso (9), si es necesario, y tomando la información necesaria (10).
- El método *Service* utiliza los métodos del objeto *Response* (11) para enviar la respuesta al servidor (12) y de ahí al cliente (13).

GET es el método que se utiliza siempre que la información se envía mediante la propia dirección *URL*, o cuando así se indica en un formulario haciendo que el atributo *METHOD* de la etiqueta *FORM* tome el valor *"GET"*. El método *POST* se utiliza exclusivamente en los formularios y se indica igualmente mediante el atributo *METHOD* de la etiqueta *FORM*.

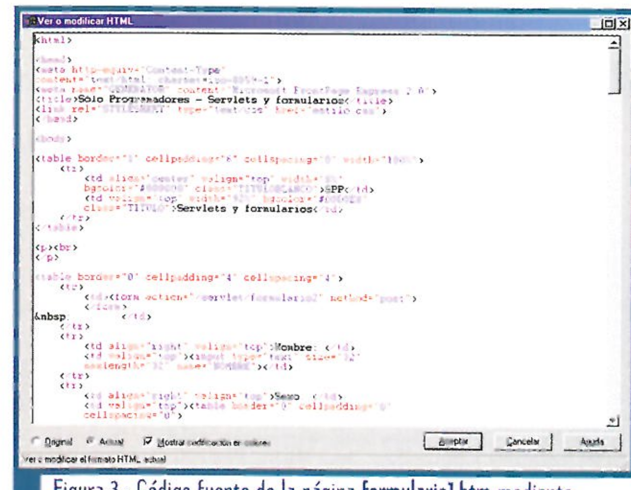
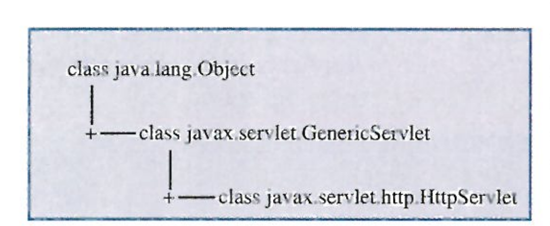


Figura 3.- Código fuente de la página **formulario1.htm** mediante la cual se puede remitir un formulario a un *servlet*.

LOS SERVLETS Y EL PROTOCOLO HTTP

Tal y como hemos adelantado, la clase *HttpServlet* es una subclase de la clase *GenericServlet* que se ha especializado fundamentalmente para manejar todos los detalles del protocolo *HTTP*, que es el que actualmente da soporte a la práctica totalidad de las aplicaciones *Web*.



En la clase *HttpServlet* el método *service* no es abstracto, tal y como ocurre en la clase *GenericServlet*. Cuando el servidor llama al método *service* de la clase *HttpServlet*, determina si el método de envío, lo que se conoce como *request method*, ha sido *GET* o *POST* e invoca al método que corresponda, *doGet* o *doPost* respectivamente.

El método getParameterValues se usa para parámetros de selección múltiple

La diferencia que existe entre ambos métodos, más allá de lo explicado, reside en el hecho de que los datos enviados mediante el método *GET* utilizan variables de entorno mientras que los enviados mediante el método *POST* se transfieren al servidor a través de los canales de entrada/salida estándar. El espacio del que dispone un servidor para variables de entorno es limitado por lo que en general se recomienda utilizar el método *POST* cuando el número y tamaño de la información sea considerable.

De la misma forma, los objetos *Request* y *Response* de los que antes hemos hablado se transforman, circunscritos al protocolo *HTTP*, en los objetos *HttpServletRequest* y *HttpServeltResponse*. Estos dos objetos constituyen el vehículo a través del cual se produce la comunicación entre el cliente y el servidor.

FORMULARIOS

Los formularios *HTML* constituyen una de las formas más comunes de recabar e intercambiar información con los usuarios. Para adentrarnos en la programación de *servlets* vamos a comenzar con un ejemplo sencillo en el que un *servlet* se encarga de recoger y procesar los datos procedentes de un formulario *HTML*.

Cuando hay muchos datos es preferible el método POST

La página **formulario1.htm** contiene el código fuente de la página *HTML* que presenta el formulario que ha de rellenar el usuario. Como se puede observar se han utilizado prácticamente todos los tipos de controles posibles: un campo de texto, casillas de selección, una lista desplegable de selección única, una lista de selección múltiple y una casilla de verificación. En general para un *servlet* todos estos elementos fun-



cionan de la misma manera, pero hay que tener en cuenta algunos casos especiales que veremos a continuación.

En primer lugar, en el código del método *doPost* se recogen todos los parámetros. La variables donde se guardan estos valores son todas ellas objetos de tipo *String* excepto en el caso de la lista desplegable de selección múltiple.

```
String p_nombre = req.getParame-
    ter("NOMBRE");

String p_sexo = req.getParame-
    ter("SEXO");
String p_pais = req.getParame-
    ter("PAIS");
String[] p_aficiones = req.getPa-
    rameterValues("AFICIONES");
String p_informacion = req.getPa-
    rameter("INFORMACION");
```

En las líneas anteriores se hace uso del método *getParameter* con la excepción, anteriormente indicada, del parámetro **"AFICIONES"**. Este parámetro puede devolver varios valores ya que el usuario puede seleccionar distintos elementos de la lista. Por ello se recurre al método *getParameterValues*, el cual devuelve un *array* de objetos de tipo *String*. Habrá tantos como valores tome el parámetro.

```
if (p_nombre != null) {
  out.println("NOMBRE: " + p_nom-
    bre);
}
else {
  out.println("NOMBRE: NULL");
}
```

El código que sigue a continuación en el *servlet* presenta el mismo esquema: se comprueba si la variable tiene un valor distinto de *null*, y dependiendo de esto se muestra una información u otra. Nótese que los campos de texto no devuelven *null* incluso cuando el usuario no ha introducido nada. El valor *null* significa que ese parámetro no ha sido enviado al *servlet*, y esto ocurre, por ejemplo, cuando no se ha seleccionado nada de una lista desplegable, o cuando la casilla de verificación no ha sido activada.

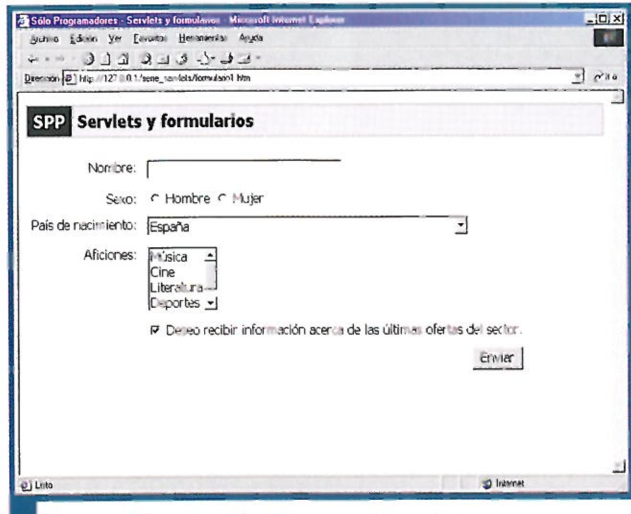


Figura 4.- Página **Formulario1.htm** con el formulario que va a ser remitido al *servlet* **Formulario2.**

La única variación sobre el bloque anterior es la que introduce el código relativo al tratamiento del parámetro múltiple **"AFICIONES"**:

```
if (p_aficiones != null) {
  out.println("AFICCIONES: ");

    for(i=0;i<p_aficiones.length;
    i++) {
    out.println(p_aficiones[i] +
    " ");
  }
} else {
  out.println("AFICCIONES:
    NULL");
}
```

En este caso se recorre el *array* **p_aficiones** y se van extrayendo los valores que toma el parámetro. Para hacer esto se utiliza la propiedad *length* de un *array*, que indica, como es evidente, el número de elementos que configuran ese *array*. Esto mismo podría aplicarse a los parámetros de valor único, pero en esos casos el *array* solamente tendría una posición.

El valor null significa que el parámetro no se ha enviado

En este ejemplo hemos visto algunos de los métodos más utilizados en los *servlets*. A continuación estudiaremos de manera formal esos y algunos otros con el fin de tener una visión clara de toda la *API* de *Java servlets*.

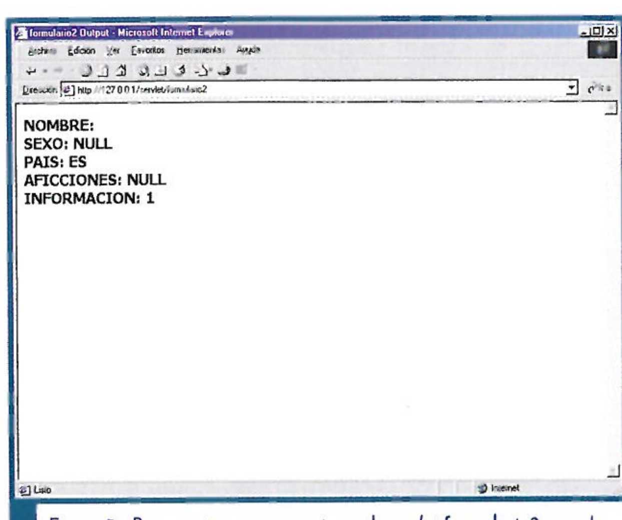


Figura 5.- Respuesta que proporciona el *servlet* **Formulario2** cuando enviamos el formulario tal cual se muestra en la pantalla.

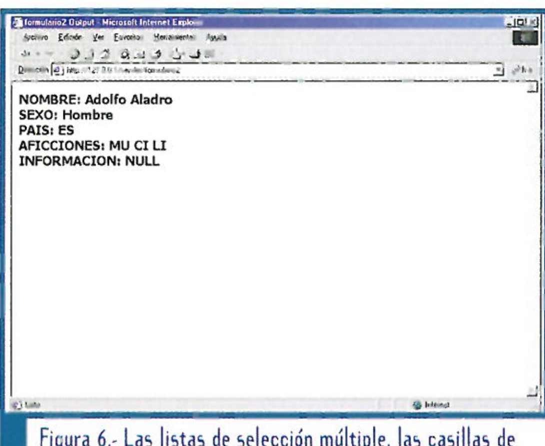


Figura 6.- Las listas de selección múltiple, las casillas de verificación y las de selección sólo se envían cuando se elige alguna opción, como se puede observar en este caso.

LOS MÉTODOS DE SERVLETREQUEST Y HTTPSERVLETREQUEST

- **Public abstract Enumeration getParameterNames()**

Devuelve los nombres de los parámetros de la petición como un objeto *Enumeration* de cadenas –objetos de tipo *String*-, o vacío si no existe ningún parámetro. Los métodos de la interfaz *Enumeration* nos permitirán recorrer la lista con todos los nombres de los parámetros.

```
for (Enumeration e =
 req.getParameterNames();e.hasMo-
    reElements();) {
   out.println(e.nextElement());
}
```

El método *hasMoreElements* devuelve **true** en el caso de que todavía queden elementos por recorrer. El método *nextElement* sirve para obtener el siguiente elemento del conjunto.

- **Public abstract String[] getParameterValues(String name)**

Devuelve todos los valores del parámetro que se pasa como argumento, como un *array* de objetos de tipo *String*, o *null* si el nombre del parámetro no existe.

- **Public abstract String getQueryString()**

Devuelve el contenido que tendría la variable de entorno *QUERY_-STRING* descrita formalmente por la interfaz *CGI*.

Como es bien sabido, una dirección *URL* puede utilizarse como vehículo para enviar información desde el navegador hasta el servidor. El formato en el que se encuentran codificados los datos consiste en un flujo de pares *nombre=valor* separados por el signo "ampersand" (&). Es decir:

```
nombre_1=valor_1&
    nombre_1=valor_1&···& nom-
    bre_n=valor_n
```

Además, la información se encuentra codificada en formato *URL* "url-codificada" (*URLencoded*), lo que significa que los espacios en blanco se sustituyen por el signo más + y los caracteres especiales se indican en hexadecimal precedidos por el signo tanto por ciento: %xx.

Según dispone la especificación de la interfaz *CGI*, en la variable de entorno *QUERY_-STRING* queda almacenada cualquier cosa que se escriba a continuación del primer signo "cierre de interrogación" ? en la dirección *URL* correspondiente.

Los primeros programas *CGI* hechos en *C* o *Perl* solían tomar esta variable, la procesaban y entonces obtenían los nombres de los parámetros así como sus valores. El entorno que ofrecen los *servlets* efectúa este proceso de manera transparente al programador por lo que no suele ser necesario conocer el valor de la variable *QUERY_STRING*. A veces puede resultar muy útil en tareas de depuración, revisión y control del código desarrollado.

La QUERY_STRING es algo heredado de la interfaz CGI

En este sentido cabe mencionar el método *encodeURL* de la clase *HttpServletResponse*, el cual sirve para codificar una dirección *URL*. Suele utilizarse con frecuencia en aquellos casos en los que un *servlet* genera una página *HTML* como respuesta y en ella existen direcciones *URL* que contienen parámetros. Nunca se debe olvidar aplicar la codificación ya que de lo contrario el funcionamiento de ese enlace no sería el deseado. Los parámetros no llegarían al servidor o lo haría con errores.

- **Public abstract String getParameter(String name)**

Devuelve el valor del parámetro cuyo nombre se pasa como argu-

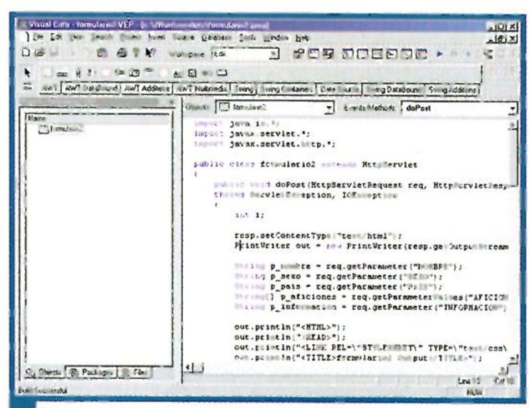


Figura 7.- Código fuente del *servlet* que procesa los datos procedentes del sumario.



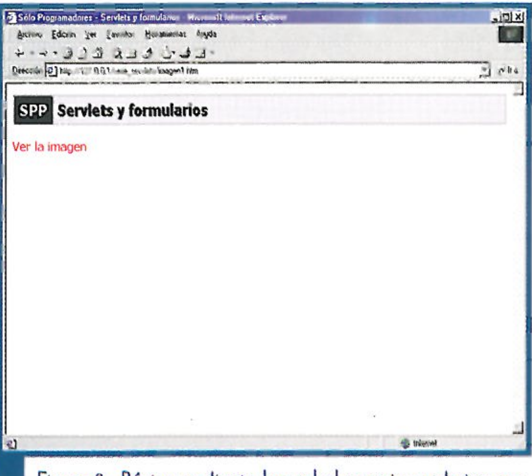


Figura 8.- Página mediante la cual el usuario puede invocar al *servlet* que devuelve una imagen.

mento, o **null** si no existe un parámetro con dicho nombre. En el caso de que se trate de un parámetro que puede tomar múltiples valores debemos utilizar *getParameterValues*.

Acabamos de ver el método *getQueryString*, el cual proporciona el valor de la variable *QUERY_STRING* descrita por la interfaz *CGI*. La *API* de *Java servlets* cuenta con algunos métodos más, gracias a los cuales podemos conocer toda aquella información que es puesta por el servidor a disposición de un programa *CGI* mediante variables de entorno, tal y como describe la interfaz *CGI*. La Tabla 1 ofrece un resumen de algunas de estas equivalencias.

LOS MÉTODOS DE SERVLETRESPONSE Y HTTPSERVLETRESPONSE

- **Public abstract ServletOutputStream getOutputStream() throws IO Exception**

Devuelve un canal de salida que se utiliza para escribir la respuesta que el servidor envía al cliente. Antes de utilizar el canal, éste puede ser convertido a otro canal de cualquier tipo aprovechando las facilidades que ofrece *Java*.

- **Public abstract void setContentLength (int len)**

Marca la cabecera *CONTENT_LENGTH* para esa respuesta, o lo que es lo mismo, indica al navegador del cliente cuál es el tamaño en *bytes* de la respuesta que va a recibir.

- **Public abstract void setContentType(String type)**

Establece la cabecera *CONTENT_TYPE* para esa respuesta, o lo que es lo mismo, indica al navegador de cliente cuál es el tipo *MIME* de la información que va a recibir. Evidentemente, este método y el anterior deben ser utilizados antes de enviar cualquier otro dato.

- **Public abstract void sendRedirect(String location) throws IOException**

Envía una respuesta al cliente que consiste en un redirección a la dirección *URL* especificada por el argumento.

Dentro del *servlet* **imagen2.java** encontramos otro ejemplo en el que se utilizan los métodos *setContentType* y *setContentLength* para devolver una imagen al navegador del cliente en vez de una página *HTML*. La estructura del *servlet* es la misma que en los ejemplos que hemos visto con la salvedad de que ahora leemos *byte* a *byte* el contenido de un archivo y lo volcamos en bloques de **1024** *bytes* por el canal de salida que proporciona el objeto de tipo *ServletOutputStream*.

SetContentType indica el tipo MIME

En primer lugar se indica al cliente cuál es el tipo de dato que va a recibir a continuación:

```
resp.setContentType("image/gif");
```

Los tipos *MIME* se expresan siempre de la misma manera, mediante el par *tipo/subtipo*. El tipo *MIME* por defecto es *text/html*, que es el que se corresponde con las páginas *HTML*.

Los datos de una dirección URL deben estar codificados

A continuación se almacena en una variable de tipo *String* la ruta al fichero que el *servlet* va a enviar al navegador del cliente como respuesta a su petición.

```
String ruta = "C:\\figura01.gif";
FileInputStream fichero = new
    FileInputStream(ruta);
```

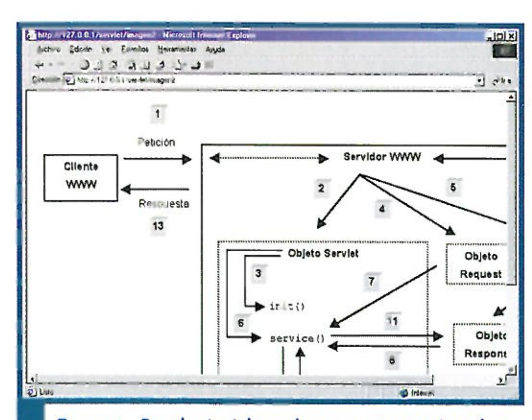


Figura 9.- Resultado del *servlet* imagen2, que devuelve una imagen en vez de una página *HTML*.

Para leer el fichero *byte* a *byte* se utiliza un canal del tipo *FileInputStream*. Para crear una instancia de este objeto se pasa como parámetro la ruta del fichero que deseamos abrir.

Los ficheros pueden leerse de varias maneras, pero para este tipo de casos, donde no se trata de un fichero de texto, conviene hacerlo en bloques. Para ello se crea un *array* de bytes de **1024** posiciones.

```
byte[] imagen_en_bytes = new
    byte[1024];
int bytes_leidos = 0;
while ((bytes_leidos =
    fichero.read(imagen_en_bytes)
    ) != -1) {
  out.write(imagen_en_bytes, 0,
    bytes_leidos);
  out.flush();
}
```

El método *read* del objeto de tipo *FileInputStream* lee del fichero bloque a bloque y devuelve el número total de *bytes* leídos. Devuelve **-1** cuando el fichero ha sido leído por completo. Cada uno de los bloques es escrito en el canal de salida mediante el método *write*. Éste recibe tres parámetros: el *array* de *bytes* que va a escribir, la posición dentro del *array* a partir del cual empezará y finalmente el número de *bytes* totales de ese *array* que se desean escribir.

Es mejor leer los ficheros binarios por bloques

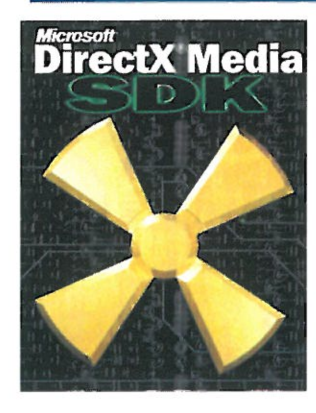
Los canales de escritura/lectura suelen tener *buffers* para incrementar la eficiencia en la transmisión, y sobre todo, para evitar problemas de sincronización. En algunas ocasiones resulta más conveniente forzar el vaciado de ese *buffer* interno después de realizar una escritura. Ésta es la finalidad del método *flush*. Cada vez que se escribe un bloque de *bytes* es mandado inmediatamente al navegador del cliente. De esta forma el usuario puede empezar a ver la imagen parcialmente desde el primer momento.

Nótese que en este ejemplo cabría plantearse si no sería mucho más óptimo tener almacenado en una variable estática, compartida por todos los *threads* del *servlet*, los *bytes* correspondientes al fichero de la imagen. Esto podría hacerse si el código encargado de leer el fichero estuviera en el método *init*. De esta forma la imagen solamente se leería una vez y las peticiones podrían atenderse con mayor rapidez al estar ésta cargada en memoria y lista para ser enviada en todo momento.

No obstante este tipo de valoraciones deben realizarse como sumo cuidado. Si se optara por guardar demasiadas cosas de manera estática, el recurso de la memoria podría verse seriamente afectado con lo que al final se dañaría el rendimiento global de la aplicación.

TABLA 1. Correspondencia entre la API Java Servlets y algunas de las variables descritas por la interfaz CGI.

Variable CGI	Método de la interfaz HttpServletRequest
REQUEST_METHOD	public abstract String getMethod() Método HTTP empleado para enviar información al servidor: normalmente GET o POST.
SCRIPT_NAME	public abstract String getServletPath() Dirección URL que identifica al Servlet.
PATH_INFO	public abstract String getPathInfo() Es posible pasar cierta información en la dirección URL de invocación del servlet. Esta información se sitúa tras el nombre del servlet en la dirección URL y está precedida por el signo de interrogación (?) que indica el comienzo de la información.
QUERY_STRING	public abstract String getQueryString() La parte que figura a partir del signo de cierre de interrogación (?) en la dirección URL del servlet.
SERVER_PROTOCOL	public abstract String getProtocol() La versión y extensión del protocolo empleado por el servidor de información. Por ejemplo, HTTP/1.0
CONTENT_LENGTH	public abstract int getContentLength() Tamaño en bytes de los datos trasnmitidos.
CONTENT_TYPE	public abstract String getContentType() Tipo MIME de los datos.
SERVER_NAME	public abstract String getServerName() Nombre del servidor tal y como se indicó en la parte host de la dirección URL de invocación del servlet. Este nombre puede ser una dirección IP o un alias de DNS.
REMOTE_ADDR	public abstract String getRemoteAddr() Dirección IP del agente que invocó al servlet.
REMOTE_HOST	public abstract String getRemoteHost() Nombre completo del agente que invocó al servlet.



Microsoft DirectX 7

Constantino Sánchez Ballesteros. *Técnico Superior Desarrollo Aplicaciones Informáticas.*

Recientemente ha llegado al usuario la última revisión de *DirectX* 7, de tal forma que este es un momento propicio para indagar en las nuevas características que se han incorporado, que no son pocas...

COMPONENTES DE DIRECTX FOUNDATION

Para los más despistados, conviene indicar que *DirectX* es la *API* de programación multimedia más avanzada y potente (más que nada porque no hay otra que actualmente pueda hacerle sombra). A continuación se detallan los componentes que forman *DirectX Foundation*:

- **Direct3D:** es la parte encargada de la representación de entornos *3D*. Existen dos versiones de esta *API*: el modo Retenido y el modo Inmediato. El Retenido es una interfaz de alto nivel para rápidos desarrollos de componentes *3D*. El modo Inmediato es muchísimo más potente pero requiere mayor esfuerzo y conocimiento en la realización de programas que representen gráficos tridimensionales. El modo Retenido en sí no forma parte de *DirectX Foundation*, sino de *DirectX Media SDK*.

- **DirectDraw:** acelera técnicas de animación por *hardware* y *software* permitiendo acceso directo a los *bitmaps* establecidos en *buffers* de memoria. Tiene un *blitting* (para representación de *sprites*) muy rápido y un volcado de *buffers* (*buffer flipping*) efectuado por *hardware*, si éste se encuentra disponible.

- **DirectInput:** se encarga del soporte de la mayoría de periféricos de entrada (ratón, *joystick*, teclado, etc.) y dispositivos *force-feedback*.

- **DirectMusic:** es la *API* de más reciente creación. Se utiliza para reproducir archivos musicales y efectos de sonido *DLS* nivel 2 con composición y variación en tiempo de ejecución.

- **DirectPlay:** Se encarga del soporte de red. Permite la conexión de juegos sobre un *módem* o una red de forma fácil.

- **DirectSetup:** se utiliza para realizar programas de instalación para *DirectX 7*.

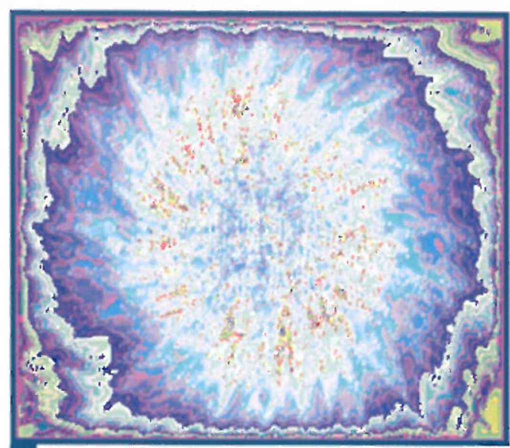


Figura 1.- *DirectDraw* funcionando bajo *Visual Basic*.



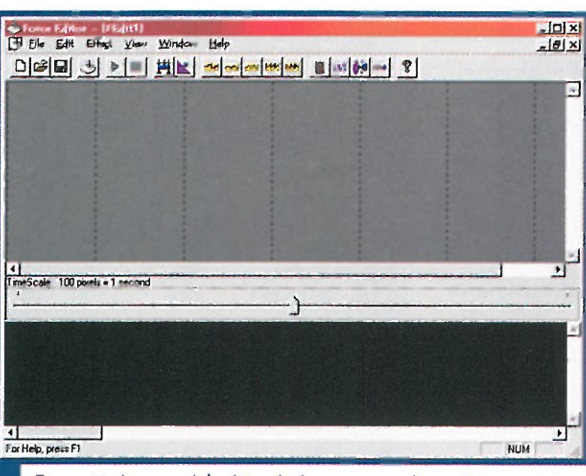


Figura 2.- Imagen del editor de fuerzas para dispositivos *ForceFeedback*.

- **DirectSound:** permite la reproducción de sonidos *Wave* (*.WAV*) e incluye soporte para el *Mixing* (mezclado) por *hardware* o *software*, posicionamiento *3D* y captura de sonido.

NUEVAS CARACTERÍSTICAS

A continuación se detallan las nuevas características de la *API* de programación multimedia más conocida:

- Una de las novedades más notables es el soporte para *Visual Basic*. La nueva librería de tipos permite que *DirectX* se desarrolle mediante este conocido lenguaje de programación.

- El Modo Inmediato de *DirectX* ofrece nuevas características como mapeado de entornos cúbicos (*cubic environment mapping*), planos de *clipping* definidos por el usuario y otros apuntes que veremos con detalle en el siguiente apartado.

- Una nueva librería de utilidades (*Direct3DX*) que simplifica las tareas más comunes que puede realizar un desarrollador de gráficos en *3D*.

- *DirectMusic* permite el soporte de *DSL2* (nivel 2 para descarga de sonidos), formato común para las nuevas tarjetas de sonido como *Sound Blaster Live*!

- *DirectInput* ahora soporta ratones de hasta ¡8 botones!, acceso exclusivo al teclado y pausas en el comienzo de efectos de los dispositivos *force-feedback* (encontrados en *joysticks* y volantes especiales). Se han implementado nuevos métodos que permiten la lectura y escritura de archivos con efectos *force-feedback* predefinidos, y un programa editor de fuerza para la creación de dichos efectos.

- *DirectSound* implementa un nuevo controlador de voz por *hardware*.

- Nuevos algoritmos de procesamiento de audio *3D* permiten una mejor utilización y rendimiento de la *CPU*.

CAMBIOS EN EL MODO INMEDIATO

DirectX 7 mantiene compatibilidad hacia atrás de todos los métodos e interfaces creados anteriormente. Además, se han implementado dos nuevas interfaces de *Direct3D*:

- *IDirect3D7*
- *IDirect3DDevice7*

Estas interfaces no tienen mucho en común con sus ancestros de versiones previas. Estos nuevos objetos permiten nuevas características, mejor rendimiento y sobre todo, facilidad de uso.

NUEVAS IMPLEMENTACIONES

- **Luces y transformación con aceleración por hardware:** *Direct3D* posibilita, con las tarjetas *3D*, acelerar las operaciones de transformación y luces por *hardware*.

- **Environment mapping con cubic environment maps:** *Direct3D* y *DirectDraw* incluyen soporte de un tipo de mapa de texturas especial utilizado en mapeados de entorno: *cubic environment map*, que utiliza una textura de seis caras que puede contener imágenes aplicadas a objetos en una escena.

- **Cambios en la API:** *Direct3D* ha cambiado el sistema para las luces, materiales y *viewports*. Ahora, éstos han sido movidos a la interfaz del dispositivo (*Device*).

- **Geometry Blending:** En *Direct3D* se ha añadido este nuevo soporte. *Geometry blending* puede ser utilizado para crear efectos de piel e incrementar el realismo de los objetos segmentados en una escena, especialmente caracteres.

- **Gestión de texturas:** El *manager* de texturas de *Direct3D* se ha ampliado para permitir a las aplicaciones

efectuar prioridades sobre el manejo de texturas. *Direct3D* utiliza estas prioridades para determinar qué texturas se mantendrán en memoria y cuáles se eliminarán.

- **Soporte para Visual Basic:** Al igual que los demás componentes de *DirectX 7*, el Modo Inmediato de *Direct3D* expone sus servicios para el entorno *Visual Basic*. Ésta es una de las características más requeridas por la mayoría de los programadores que odian la complejidad de *C++*.

- **Emulación por software ampliada:** *Direct3D* ha sido optimizado para utilizar instrucciones especiales soportadas por la *CPU*. Entre éstas se incluyen las *3D-Now*! de *AMD* y las *MMX* de *Intel*. Siempre que sea posible, *Direct3D* utilizará *3D-Now!* para acelerar la transformación y luces y *MMX* se utilizará para acelerar el proceso de *raster*.

EJEMPLO EN VISUAL BASIC (DIRECTDRAW)

Para probar alguna de las nuevas características de *DirectX 7*, vamos a utilizar el nuevo soporte para *Visual Basic* que implementa la *API*. Cuando programemos bajo este entorno de trabajo, debemos pensar que la esencia de *DirectX* no cambia en relación con *C++*. Las funciones son las mismas, lo único que cambia es la forma en que definimos variables, creamos funciones y procedimientos, es decir, cosas específicas para cada lenguaje de programación.

El programa que vamos a crear realizará una tarea simple, pero bastante descriptiva de cómo utilizar *DirectX* en *Visual Basic*. Esta tarea consistirá en visualizar en pantalla un gráfico de un tigre, incluido en el *SDK* de *DirectX*. Por supuesto, el usuario podrá cambiar el gráfico utilizando uno de su propia cosecha. Para ello, sólo habrá que cambiar un parámetro de un método que veremos más adelante.

Utilizaremos un formulario con su correspondiente módulo de código. El primer paso en la creación de la aplicación consiste en definir las variables que se utilizarán como globales para todo el módulo.

DirectX 7 es la última revisión de esta API multimedia

El *Clipper*, que se utiliza para visualizar *DirectX* en ventana normal, se enlazará a un control *PictureBox* que hará las veces de formulario inicial para la representación gráfica. Es necesario definir variables globales, como mínimo, para los siguientes objetos:

- *DirectX*
- *DirectDraw*
- *DirectDrawSurface*
- *DirectDrawClipper*
- *DirectDrawPalette*

Otras son de menor importancia pero se utilizan para establecer las características de los *buffers* gráficos, etc.

```
Option Explicit

Dim objDX As New DirectX7
Dim objDD As DirectDraw7
Dim objDDSurf As
    DirectDrawSurface7
Dim objDDPrimSurf As
```



Figura 3.- Detalle de la propiedad *ScaleMode* dentro de las propiedades del formulario en *VB*.

```
    DirectDrawSurface7
Dim ddsd1 As DDSURFACEDESC2
Dim ddsd2 As DDSURFACEDESC2
Dim ddClipper As
    DirectDrawClipper

Dim bInit As Boolean
Dim pal As DirectDrawPalette
```

Cuando efectuemos la carga del formulario, llamaremos al procedimiento *init()*. Éste se encargará de inicializar los objetos de *DirectDraw*, su nivel cooperativo, la creación de *buffers*, etc.

```
Private Sub Form_Load()
    init
End Sub
```

Veamos paso a paso las tareas que se realizan dentro de este procedimiento.

```
Sub init()
```

La cadena vacía dentro del parámetro significa que utilizaremos el *Driver* de vídeo establecido por defecto. Para establecer comparaciones con *C/C++*, en éste utilizaríamos el valor *NULL*.

```
Set objDD =
    objDX.DirectDrawCreate("")
```



La aplicación utilizará una ventana al igual que cualquier aplicación de *Windows*, además de utilizar la misma profundidad de color que haya establecida. Por eso, utilizaremos la característica **DDSCL _NORMAL.**

```
Call
    objDD.SetCooperativeLevel(Me.
    hWnd, DDSCL_NORMAL)
```

Debemos indicar que el miembro *ddsCaps* es válido para este tipo.

```
ddsd1.lFlags = DDSD_CAPS
```

El editor de texturas permite crear mipmaps y conversiones a formatos DXTn

Es el momento de crear el *buffer* primario, que representará la pantalla. Este *buffer* será el que se visualizará en el monitor cada vez que se hagan operaciones gráficas.

```
ddsd1.ddsCaps.lCaps =
    DDSCAPS_PRIMARYSURFACE
```

El siguiente paso es crear el *buffer* primario con las características que hemos definido antes.

```
Set objDDPrimSurf =
    objDD.CreateSurface(ddsd1)
```

Ahora estableceremos las características para el segundo *buffer* que utilizaremos como paso intermedio para representar los gráficos antes de que salgan por pantalla.

```
ddsd2.lFlags = DDSD_CAPS
ddsd2.ddsCaps.lCaps =
    DDSCAPS_OFFSCREENPLAIN
```

Creamos el *buffer* secundario cargando en él el gráfico que deseemos, en este caso, **tigre.bmp:**

```
Set
    objDDSurf=objDD.CreateSurface
    FromFile("tigre.bmp",
    ddsd2)
```

El *Clipper* servirá para visualizar el *buffer* primario en una ventana normal de *Windows*.

```
    Set ddClipper =
    objDD.CreateClipper(0)
    ddClipper.SetHWnd
    Picture1.hWnd
    objDDPrimSurf.SetClipper
    ddClipper

    bInit = True
```

Ahora ya podemos volcar el gráfico, para lo que utilizaremos el procedimiento **blt.** Conviene recordar recuerdo que este procedimiento se ha definido para simplificar la claridad y reutilización del código. Más adelante veremos cómo está construido:

```
    blt
End Sub
```

Puesto que el usuario podrá redimensionar a su antojo el tamaño de la ventana, debemos preveerlo utilizando *Form_Resize*. Este procedimiento es llamado por **me.show** o cuando se redimensiona la ventana en tiempo de ejecución. Dado que *DirectX* utiliza *pixels* y *VB* utiliza *twips*, tendremos que sincronizar las dos escalas.

Cualquier programa en C/C++ que utilice DirectX, podrá ser realizado en Visual Basic

Es necesario cambiar la propiedad *ScaleMode* en el formulario al valor **Pixels**. El ancho y alto del formulario permanecerá en *twips* hasta que cambiemos el *ScaleMode*, pero *ScaleWidth* y *ScaleHeight* estarán ahora en *pixels*.

```
Private Sub Form_Resize()

    Picture1.Width =
    Me.ScaleWidth
    Picture1.Height =
    Me.ScaleHeight
    blt
End Sub
```

La librería de tipos para Visual Basic es una de las novedades más solicitadas

Es el momento de crear el procedimiento *blt*, anteriormente comentado en la inicialización del formulario. Su objetivo es transferir los datos gráficos del *buffer* secundario al primario.

```
Sub blt()
```

La variable *bInit* indica si se ha inicializado la aplicación. En caso contrario, salimos del procedimiento.

```
If bInit =
False Then
Exit Sub

Dim ddrval
As Long
Dim r1 As
RECT
Dim r2 As RECT
```

Obtenemos las coordenadas y tamaño del formulario y guardamos los valores obtenidos dentro de **r1** para su posterior utilización en el volcado gráfico.

```
Call
objDX.GetWindowRect(Picture1.
hWnd, r1)

r2.Bottom = ddsd2.lHeight
r2.Right = ddsd2.lWidth
```

El método *blt* del *buffer* primario nos permitirá que se vuelque la imagen dentro de los límites de la ventana y se visualice en la pantalla.

```
    ddrval =
objDDPrimSurf.blt(r1,
objDDSurf, r2, DDBLT_WAIT)

End Sub
```

El siguiente procedimiento es llamado en tiempo de ejecución cuando el formulario es movido o redimensionado.

```
Private Sub Picture1_Paint()
    objDD.RestoreAllSurfaces
    init
    blt
End Sub
```

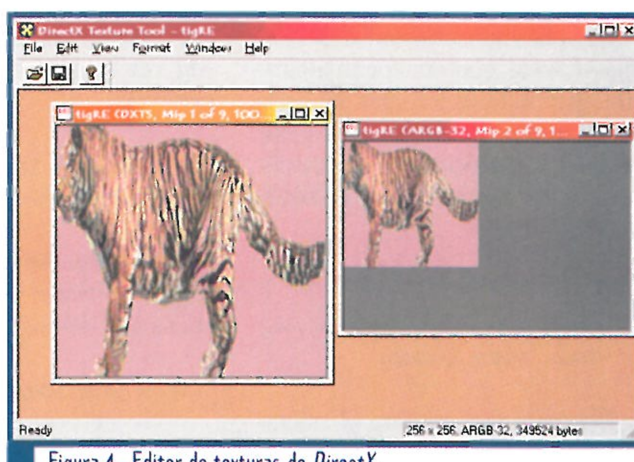


Figura 4.- Editor de texturas de *DirectX*.

DIRECTX TEXTURE EDITOR

Una de las herramientas incluidas en esta última revisión de *DirectX* es el editor de texturas. El objetivo de esta herramienta es permitir a los desarrolladores crear mapas de texturas que utilicen los nuevos formatos de compresión *DXTn*. Crear una textura comprimida no es tan complicado de hacer: el método *Blt* de *IDirectDrawSurface7* puede hacer la conversión por nosotros.

Probablemente, los desarrolladores avanzados querrán escribir sus propias herramientas que se adapten a sus necesidades, pero esta herramienta en concreto permite la siguiente funcionalidad:

- Abre archivos *BMP* y *DDS*. Éste último y nuevo formato encapsula la información en un *buffer DirectDrawSurface*. Los datos pueden ser leídos directamente dentro del *buffer* que utilicen.
- Abre archivos *BMP* como *alpha channel* (canales *alpha*, para establecer el grado de transparencia).



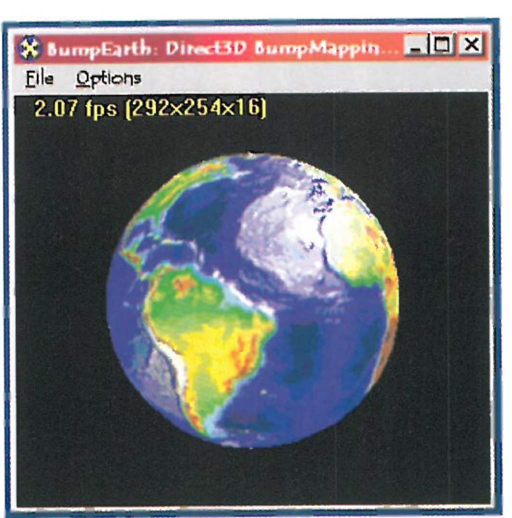


Figura 5.- El *BumpMapping* puede representarse utilizando *Direct3D*.

- Guarda texturas en formato *DDS*.
- Soporta conversiones para todos los formatos de compresión *DXTn*.
- Soporta generación de *mipmaps*.
- Soporta visualización del canal *alpha* como una imagen a escala de grises o por un color seleccionable por el usuario.
- Soporta una comparación visual inmediata (calidad de texturas) entre las imágenes utilizadas.

UTILIZACIÓN

DirectX Texture utiliza una interfaz multidocumento. Esto significa que cada mapa de texturas es un documento, y por lo tanto, múltiples documentos pueden ser abiertos al mismo tiempo.

Cada documento puede albergar una textura en uno o dos formatos al mismo tiempo. Por ejemplo, podríamos importar un *BMP* que automáticamente se convertiría en una textura de 32 *bits* de tipo *ARGB* (imagen con canal *alpha*). Seguidamente podríamos convertir la textura al formato *DXT1*. En este punto, el documento recordaría la imagen en ambos formatos y el usuario podría realizar volcados de ventana entre formatos pulsando en la ventana correspondiente.

Con este sistema podremos comprobar si hay errores al comprimir los archivos *bitmaps* y no llevarnos sorpresas a la hora de ejecutar el programa que estemos realizando. También podemos determinar qué formato presenta más calidad en la compresión.

Para crear *mipmaps* (diferentes resoluciones para una misma imagen) elegiremos la opción **Generate MipMaps** del menú **Format**. El ancho y alto de la imagen original deben ser múltiplos de 2.

El editor también soporta canal *alpha* para las texturas. Cuando importamos un archivo *BMP*, si existen dos archivos y uno tiene la coletilla "_a" al final del nombre (por ejemplo, *textura.bmp* y *textura_a.bmp*), este segundo archivo se cargará automáticamente como canal *alpha*.

Para poder ver el canal *alpha* directamente sin los canales *RGB*, elegiremos **Alpha Channel Only** del menú **View**. El canal *alpha* aparecerá como una imagen a escala de grises.

El editor también permite que se le pasen parámetros desde la línea de comandos. Podemos pasar archivos de entrada, un archivo de salida y las opciones de procesamiento que queramos efectuar sobre los archivos gráficos.

En el siguiente ejemplo se puede apreciar el detalle de parámetros desde la línea de comandos:

```
DXTEX [infilename] [-a alphaname]
    [-m] [DXT1| DXT2| DXT3| DXT4|
    DXT5] [outfilename]
```

- **Infilename:** nombre del archivo a cargar. Puede ser un *bmp* o un *dds*.
- **-a alphaname:** si se especifica "-a", el siguiente parámetro es el nombre del archivo *bmp* que puede cargarse como canal *alpha*. Si no se especifica ningún archivo *alpha*, la aplicación buscará archivos con la coletilla "_a" al final de su nombre (*infiename_a.bmp*). Si existe alguno, lo utilizará como canal *alpha*.
- **-m:** esta opción habilita la generación de *mipmaps*.
- **DXT1 | DXT2 | DXT3 | DXT4 | DXT5:** especifica el formato de compresión. Si no se especifica ningún formato la imagen se habilitará como *ARGB-8888*.
- **outfilename:** especifica el nombre del archivo de salida. Si no lo hemos declarado, la interfaz de usuario mostrará el archivo actual y todas las operaciones requeridas. En caso contrario, la aplicación saldrá después de guardar el archivo procesado y no presentará ninguna interfaz de usuario.

En la próxima revisión de *DirectX Microsoft* ha prometido mayores avances, sobre todo en representación *3D*, pero será cuando salga a la luz **Fahrenheit** cuando se vea el máximo potencial que nos puede proporcionar un *PC* en relación con la representación de entornos virtuales en tres dimensiones.













BOLETÍN DE PEDIDOS

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L.
(Revista Sólo Programadores). C/ San Sotero, 5. 1ª Planta. 28037 Madrid
Tlf: 91 304 87 64. Fax: 91 327 13 03

Deseo que me manden los número/s.................................
....
....

NOMBRE Y APELLIDOS:...
....
EDAD:........... PROFESIÓN: ....TFNO: ....
DOMICILIO: ....
CIUDAD: ....
PROVINCIA: ....C.P: ....

❑ **Precio por unidad: 975 pesetas** + 700 ptas. de gastos de envío.

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
❑ Domiciliación bancaria
❑ Contra reembolso

DATOS DE DOMICILIACIÓN:

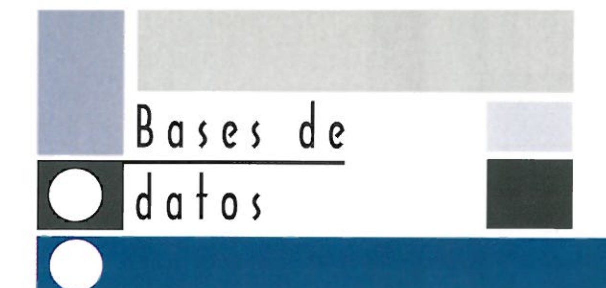
Banco.. FIRMA

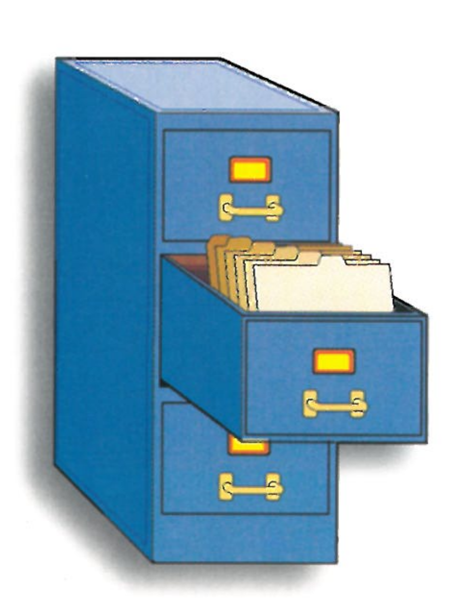
Domicilio..

Nº de Cuenta.........|.........|.........|.....................

Titular..

Fecha...





Aplicación de bases de datos con Delphi 5 (I)

Juan Luis Ceada Ramos.
Programador en ARCABE Formación y Servicios Informáticos

Comenzamos este mes una nueva serie de artículos dedicados al desarrollo de aplicaciones que trabajan con diferentes bases de datos. En cualquier caso, nos centraremos sobre todo en aquellas que ofrece *Delphi* 5.

INTRODUCCIÓN

Desde sus primeras versiones, *Delphi* ha estado enfocado al desarrollo de aplicaciones para la gestión y mantenimiento de bases de datos. Cuando apareció la primera versión, muchas empresas lo adoptaron como herramienta de desarrollo debido a la facilidad con que se crean las aplicaciones de gestión, así como a la potencia de los componentes de acceso a bases de datos y al excelente *Borland DataBase Engine* (*BDE*).

Borland decidió dividir el producto en tres versiones. La más básica, la versión estándar, sólo permite trabajar con bases de datos locales (como *Paradox* y/o *Dbase*).

Delphi está especialmente dotado para trabajar con múltiples bases de datos

La versión profesional avanza un poco más en el desarrollo de aplicaciones de bases de datos. Permite enlazar con Sistemas Gestores de Bases de Datos (en adelante, *SGBD*), como pueden ser *Oracle* o *InterBase*, ya sea de forma nativa, o mediante *ODBC*. Además, añade nuevos componentes para el acceso a bases de datos.

NOTA: Las siglas ODBC corresponden al estándar creado por Microsoft que permite a cualquier aplicación acceder a



Figura 1.- Imagen de las tres versiones de *Delphi* 5.



aquellos gestores de bases de datos que proporcionen un driver ODBC.

Es posible descargar una versión de evaluación de Delphi 5 Enterprise

Por último, la versión *Cliente/Servidor* (en *Delphi* 5, versión *Enterprise*), está más enfocada aún que la versión profesional al desarrollo en entornos *C/S*. Para ello proporciona nuevos componentes de acceso a datos (como los *TClientDataSet*), de visualización de datos, de desarrollo de aplicaciones *multi-tier* (*MIDAS*), etc. Además, incluye nuevos enlaces con diferentes *SGBD's* y nuevas herramientas diseñadas para facilitar el trabajo en entornos Cliente/Servidor (como por ejemplo, *SQL Explorer*, *SQL Monitor*, *SQL Builder*.) Para más información acerca de las capacidades de cada una de las distintas versiones, consulte la página http://www.inprise.com/delphi/productinfo/featurelist/.

Para poder seguir está serie de artículos, sería deseable contar con la versión *Enterprise*. Aquellos lectores interesados en seguir esta serie de artículos que no dispongan de dicha versión, pueden obtener una copia de evaluación en la siguiente dirección: http://www.inprise.com/delphi/trial5/trialdownload.html.

CONOCIMIENTOS MÍNIMOS

El lector debería tener unos conocimientos mínimos sobre bases de datos: qué es una base de datos, qué es una tabla, índices, integridad referencial, el modelo relacional, qué son las transacciones, restricciones, dominios, etc. Cualquier libro de bases de datos genérico (no necesariamente de *Delphi*) puede servir para adquirir estos conocimientos.

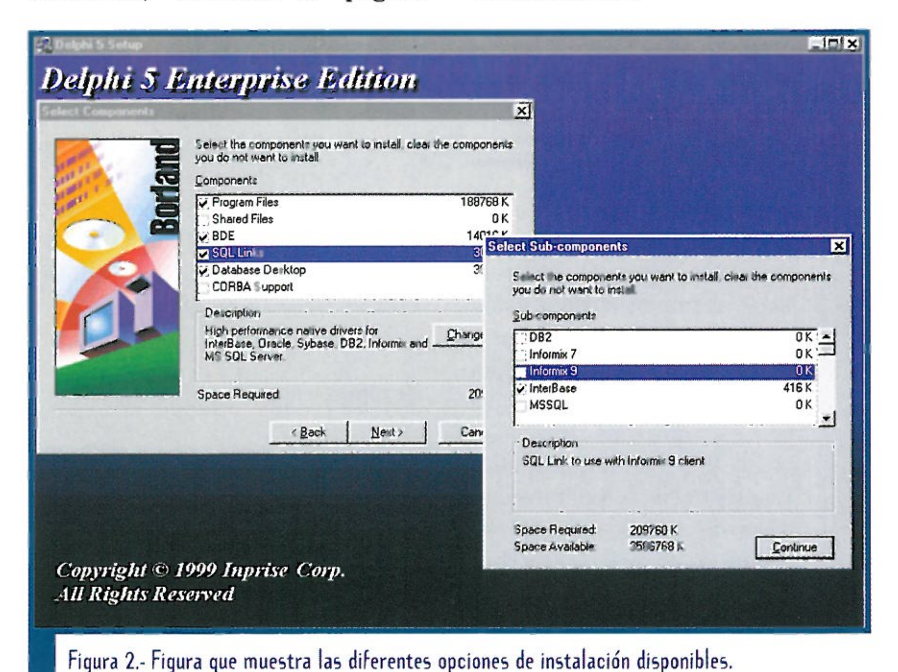


Figura 2.- Figura que muestra las diferentes opciones de instalación disponibles.

CONTENIDO DE LOS ARTÍCULOS

A lo largo de esta serie de artículos, abordaremos, entre otros, los siguientes temas:

- Conceptos básicos sobre bases de datos.
- Herramientas para trabajar con bases de datos.
- El motor de bases de datos de *Borland* (*BDE*).
- Componentes *BDE* para el acceso e introducción de datos.
- Módulos de datos.
- Ejemplos usando bases de datos *Paradox*.
- Nociones de *SQL*.
- Componentes *BDE* para trabajar con *SQL*.
- Acceso a *SGBD's* (conexión nativa y mediante *ODBC*: *Access*, *Oracle* 8, *InterBase*).
- Ejemplos usando bases de datos remotas.
- Novedades *Delphi* 5: *ADO*, Componentes *IBX*, etc.
- Conceptos Avanzados: *TClientDataSet*, *CacheUpdates*, etc.

Los temas específicos de *Delphi* 5 se presentarán al final de la serie. Para el resto de los capítulos, en principio bastará con contar con una copia de *Delphi* 3 o superior.

INSTALACIÓN DE DELPHI

En las figuras 2 y 3 se muestran las opciones que habrá que marcar durante la instalación para que dispongamos de todos los elementos necesarios para poder seguir la serie de artículos.

En la parte izquierda de la Figura 2 se observan los *SQL Links* mínimos que deberemos instalar. En nuestro caso, bastará con marcar el de *InterBase* y *Oracle*. Como se puede observar, podemos conectar de forma nativa con muchos otros *SGBD's*, entre ellos *SQL Server* de *Microsoft*.

Para seguir la serie, habrá que instalar los SQL Links de Oracle e InterBase

En un determinado punto de la instalación aparecerá un cuadro de diálogo (ver Figura 3) que permite seleccionar algunos aspectos relacionados con las versiones de *SGDB* con los que podemos conectar. En este caso, se ha optado por enlazar con *Oracle* 8 y *Access* 97. Realmente esto no es demasiado importante, puesto que podremos cambiar estos parámetros más tarde, mediante el *BDE Configurator*.

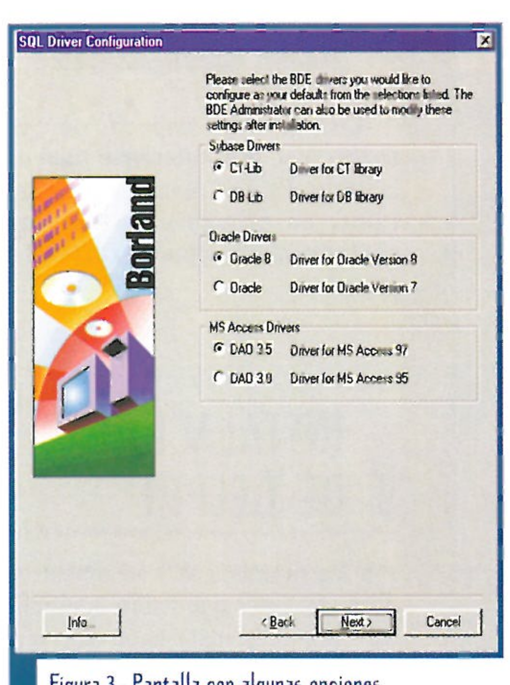


Figura 3.- Pantalla con algunas opciones de instalación adicionales.

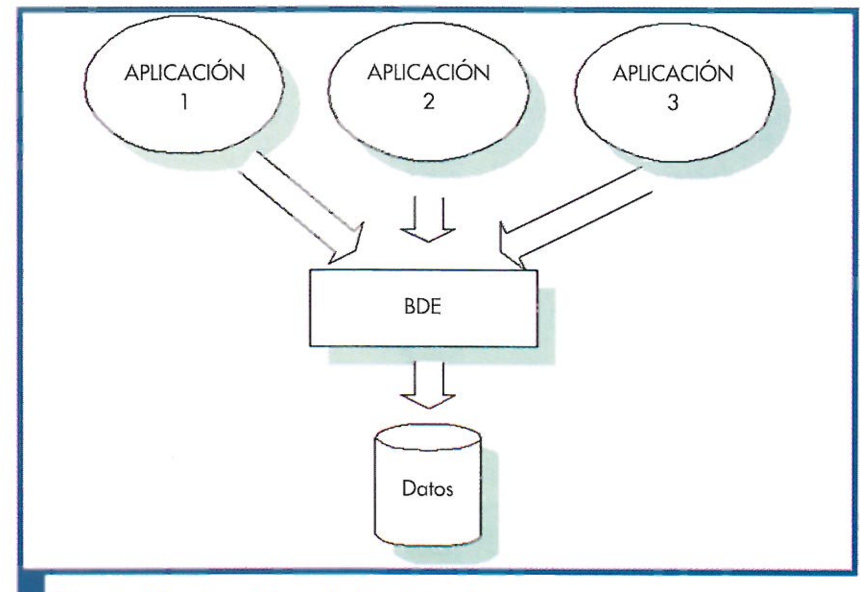


Figura 4.- Aplicaciones de bases de datos de una única capa.

BASES DE DATOS Y DELPHI

Como hemos comentado, *Delphi* está especialmente dotado para trabajar con bases de datos de muy diversos tipos. Desde bases de datos locales, hasta servidores *SQL*, en configuraciones de una, dos, tres o más capas.

BD'S LOCALES MONOUSUARIO

En este caso, todo el sistema se sustenta en una sola capa (ver Figura 4). Es decir, en el mismo ordenador están situadas las bases de datos y las aplicaciones que trabajan con ellas. Además, dichas bases de datos se almacenan como ficheros independientes. Es decir, si tenemos una tabla de clientes y otra de artículos, en el disco tendremos dos ficheros (por ejemplo, **clientes.db** y **articulos.db** si trabajamos con *Paradox*).

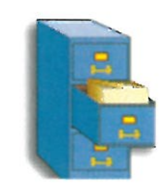
Paradox y *Dbase* son sistemas de *BD's* locales. En la terminología de *Borland*, se conocen como bases de datos de escritorio.

BD'S LOCALES MULTIUSUARIO

En realidad se trata de una ampliación del caso anterior, sólo que en este caso es la propia aplicación la que controla el acceso concurrente a las tablas. Antiguamente esta tarea debía controlarla el propio programador, que más o menos solucionaba el tema a base de insertar bloqueos. En nuestro caso, es el motor de bases de datos (el famoso *BDE*) el que se encarga de asegurar que varios usuarios pueden acceder a los datos de forma concurrente sin problemas.

Veremos cómo trabajar con BD's de escritorio y servidores SQL

Por supuesto, para que el sistema funcione, el *BDE* tiene que estar bien configurado. Es frecuente ver en los foros de noticias a gente preguntado sobre problemas con el



acceso concurrente a tablas *Paradox*. En la mayoría de los casos, es un problema de configuración. En http://www.clubdelphi.com/articulo/006/004.htm podéis leer un artículo de cómo configurar el *BDE* para acceder a tablas *Paradox* en red.

SERVIDORES SQL

En este caso, el sistema se distribuye en dos capas (ver Figura 5). En una capa se sitúan las aplicaciones, mientras que en la otra se encuentra el sistema gestor de bases de datos (un servidor *SQL*), que puede estar en la misma máquina o en una máquina remota.

Si muchos usuarios van a trabajar con los mismos datos, deberíamos optar por un servidor SQL

En este caso, las aplicaciones no acceden directamente a los datos, sino que a través del *BDE* se envían peticiones directamente al servidor *SQL*, que es el encargado de trabajar con los mismos. Puesto que sólo una aplicación accede realmente a los datos, es posible implementar en estos sistemas capacidades avanzadas, como el acceso multiusuario, transacciones, recuperación ante fallos, etc. Además, y esta es una de sus mayores ventajas, podemos incluir código en el propio servidor *SQL* (mediante *triggers* y/o procedimientos almacenados).

Los servidores SQL permiten incluir código dentro de la propia base de datos

CONFIGURACIONES MULTICAPA

En este caso, se suele trabajar con tres capas (ver Figura 6):

- En la primera se sitúan las aplicaciones clientes.
- En la segunda, se sitúa la aplicación servidora.
- En la tercera, se sitúa el *SGBD*.

La aplicación cliente hace peticiones a la aplicación servidora, siendo esta última la que transmite dicha petición al *SGBD*. Después de efectuar el proceso correspondiente con la petición, el *SGBD* devuelve los resultados al servidor, que a su vez se los pasa al cliente.

Las aplicaciones multicapa permiten aislar a los clientes de las bases de datos

La mayor ventaja de este sistema reside en que el cliente se simplifica muchísimo, facilitando de esta forma el desarrollo de clientes multiplataforma (que no son más que aplicaciones "tontas" que se limitan prácticamente a visualizar los datos).

EL MOTOR DE BASES DE DATOS

Para acceder a los datos *Delphi* utiliza el *BDE* (*Borland Database Engine*). Básicamente, se trata de un conjunto de funciones diseñadas para acceder a los datos contenidos en tablas o bases de datos. El programador no trabaja directamente con estas funciones, sino que usa componentes de la *VCL* para acceder a los datos (ver Figura 7).

El BDE se encarga de proporcionar acceso transparente a los datos

El *BDE* permite a las aplicaciones acceder de forma totalmente transparente a los datos, de tal forma que si la aplicación está bien diseñada, en teoría sería posible usar tablas *Paradox* o *Dbase*, o acceder a bases de datos en servidores *Oracle*, *InterBase*, *SQL Server*, etc, sin tener que realizar ni un sólo cambio. En la práctica, esto no siempre es posible. Volveremos

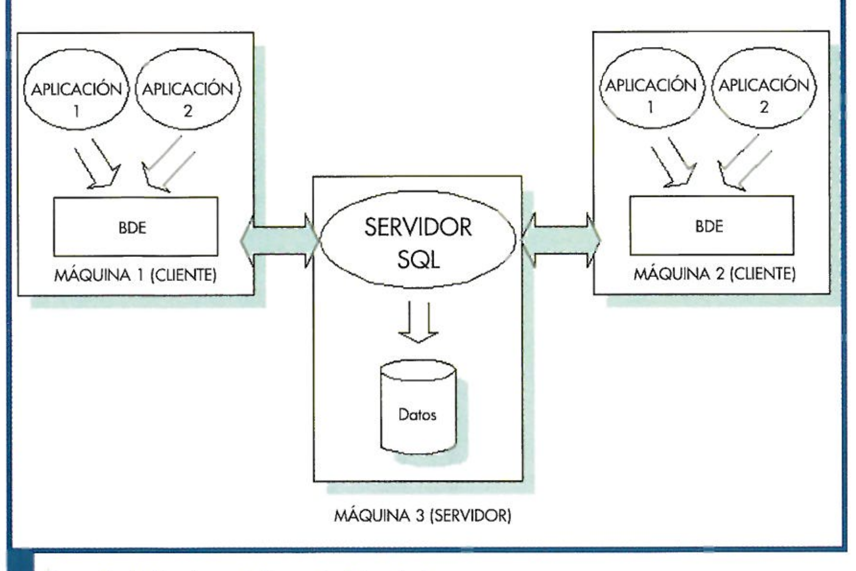


Figura 5.- Aplicaciones de bases de datos de dos capas.

sobre este punto en próximas entregas, cuando abordemos el tema de los alias.

El *BDE* proporciona además un interprete local de *SQL*, para dar soporte de este lenguaje a las aplicaciones basadas en bases de datos de escritorio. También permite implementar transacciones en sistemas que no las soportan (como *Paradox*). Por supuesto, tanto el intérprete *SQL* como las transacciones en bases de datos de escritorio tienen sus limitaciones.

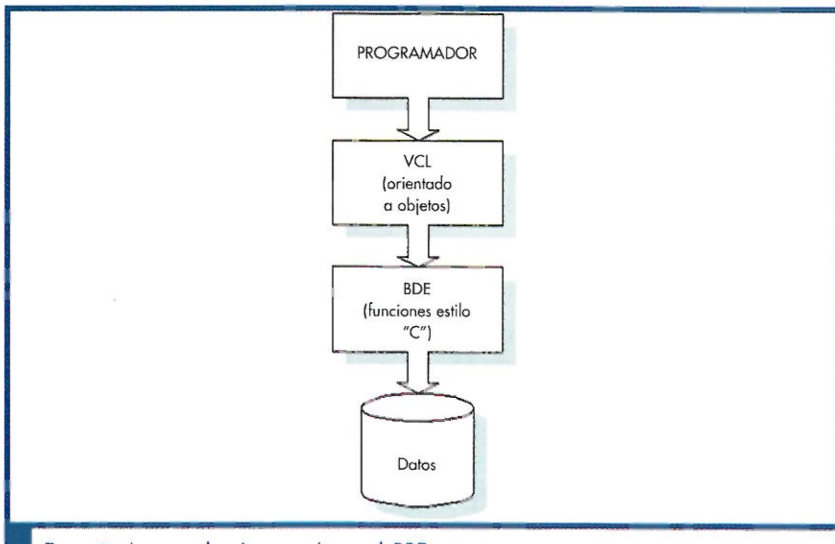


Figura 7.- Acceso a los datos mediante el *BDE*.

Gracias al BDE, una aplicación puede trabajar indistintamente con servidores SQL o BD's de escritorio

Gracias a esto es posible diseñar una aplicación que haga uso de estos recursos, y en un futuro dar el salto a un servidor *SQL* sin preocuparnos por tener que añadir código para aprovechar las nuevas características de dicho servidor.

El *BDE* ofrece una arquitectura modular. El acceso a los diferentes formatos de bases de datos se implementa en *DLL's*, lo que permite ir añadiendo soporte para los nuevos sistemas que van apareciendo sin problemas. En caso de que necesitemos acceder a un formato de *BD* que el *BDE* no soporte de forma nativa, siempre podremos acceder a través de *OBDC*.

En definitiva, el *BDE* permite que una misma aplicación pueda funcionar con bases de datos de escritorio, servidores *SQL*, con conexión nativa u *ODBC* con mínimos cambios (que en la mayoría de los casos son de configuración del propio *BDE*, y no de la aplicación en sí).

Figura 6.- Configuración multicapa.

Es posible prescindir del BDE en determinados casos

Actualmente, es posible prescindir del *BDE* en algunos casos. Para acceder a sistemas gestores de bases de datos de *Microsoft*, es posible utilizar los componentes *ADO*. Si nuestras aplicaciones trabajan únicamente con *Interbase*, podemos usar los componentes *IBX*. Hablaremos sobre estas dos alternativas en los últimos artículos de esta serie.

HERRAMIENTAS AUXILIARES

Antes de meternos de lleno a explicar cómo realizar aplicacio-



nes que accedan a bases de datos, resulta imprescindible que los lectores se familiaricen con las aplicaciones auxiliares que se distribuyen con *Delphi*. En las siguiente líneas, se indican cuáles son y para qué sirven.

BDE ADMINISTRATOR

El Administrador del Motor de Datos de *Borland* (*BDE Administrator*) permite configurar las opciones de acceso a bases de datos de *Delphi*. Podéis ver su aspecto en la Figura 8. En dicha Figura, la elipse de color azul muestra el icono asociado a los alias que usan un controlador nativo para acceder a una base de datos. La elipse roja muestra el icono que se usa cuando el acceso a la *BD* se realiza mediante *ODBC*.

El BDE Administrator permite gestionar alias y parámetros de configuración

Esta utilidad es muy importante, hasta tal punto que se instala cuando distribuimos el *BDE* con nuestras aplicaciones. Básicamente, permite realizar dos operaciones.

La primera operación, aunque será tratada con mayor detalle en entregas posteriores, se refiere a la gestión de alias, que a grandes rasgos, vienen a ser como un acceso directo a las bases de datos. Es decir, los componentes de bases de datos de *Delphi* utilizarán un determinado nombre para acceder a las tablas guardadas en disco (alias), de tal forma que es el propio alias el que apunta a un determinado disco y directorio, en el que se encuentran los datos. Por tanto, si se cambia la localización de los datos y se sitúan en otro directorio, únicamente se deberá modificar esta información en el alias utilizado, sin necesidad de cambiar ningún valor en los componentes de acceso a datos utlizados por una aplicación.

La segunda posibilidad sirve para configurar aspectos relacionados con el formato de los números y fechas (ver Figura 9), así como determinados parámetros de los controladores nativos y *ODBC* para el acceso a las bases de datos. En la mayoría de los casos, bastarán los parámetros por defecto (y en el caso de que tengamos que modificar alguno de ellos, lo podremos hacer mediante código desde nuestra aplicación).

DATABASE DESKTOP

Se trata de una pequeña utilidad que permite realizar el mantenimiento de tablas: creación de tablas (tanto en sistemas de escritorios como en sistemas *SQL*), modifición, inserción de datos, eliminación de datos, realización de consultas *SQL*, creación de alias, etc. También permite realizar otras operaciones con las tablas (como añadir datos de forma masiva, eliminarlos de forma masiva, obtener información, vaciar las tablas, etc).

Database Desktop es una buena opción para el mantenimiento de tablas en BDs de escritorios

Además, permite añadir restricciones a determinados campos, crear indices primarios y secundarios, definir la integridad referencial, añadir *passwords* a las tablas, etc.

En la Figura 10 se puede apreciar el aspecto que tiene dicha aplicación. Es sencilla de usar, pequeña, manejable y bastante intuitiva. *Borland* realizó un gran trabajo al crear está utilidad. Tan buen trabajo, que no ha necesitado ningún cambio desde que aparición en el mercado la versión 7.0, junto a *Delphi 2*.

Eso sí, aunque es posible crear tablas de cualquier tipo con esta

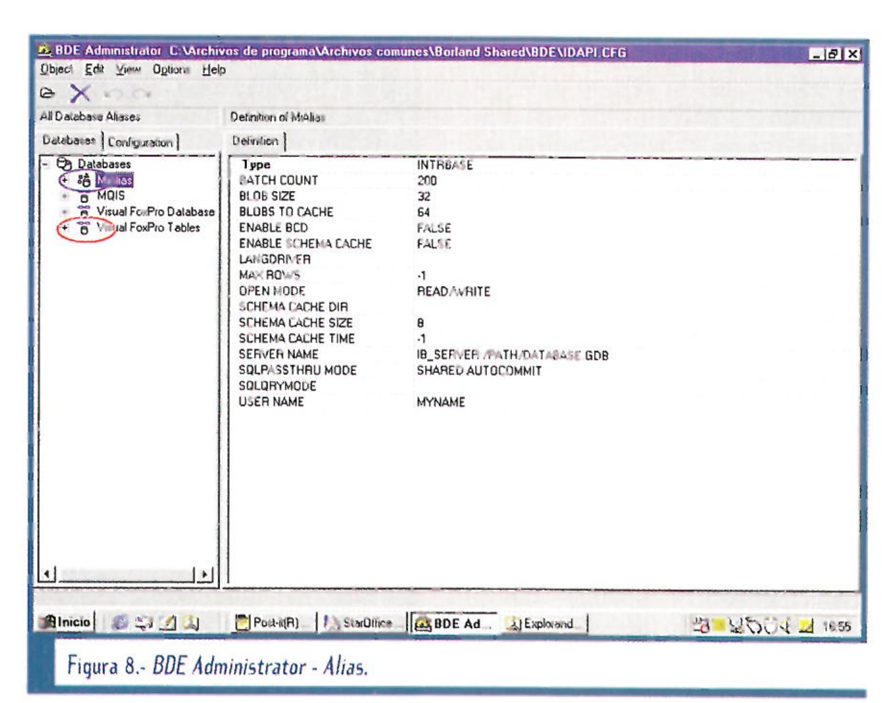


Figura 8.- *BDE Administrator - Alias.*

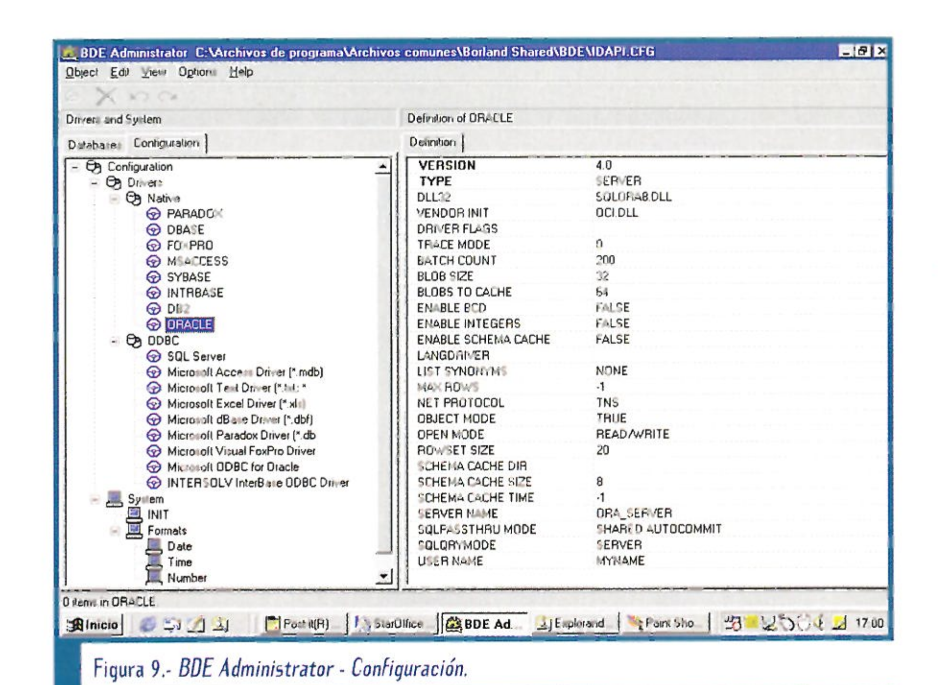


Figura 9.- *BDE Administrator - Configuración.*

aplicación, la mayoría de las veces la usaremos para crear tablas *Paradox* y/o *Dbase*.

SQL EXPLORER

En las versiones *Cliente/Servidor* de *Delphi*, se incluye está utilidad, que permite gestionar alias y parámetros de configuración, crear bases de datos, tablas, índices primarios y secundarios, introducir datos, consultar datos (mediante *SQL*), etc. Es como si mezclásemos el *BDE Configurator* con el *Database Desktop* en un único producto, funcional, y muy cómodo de usar.

SQL Explorer reúne en una sola herramienta al BDE Configurator y al Database Desktop

Con él podemos hacer todo lo que se nos ocurra, y lo más importante, al contrario que el *DataBase Desktop*, este producto sí está indicado para trabajar con cualquier tipo de base de datos, tanto las de escritorio como las alojadas en servidores *SQL*.

Otra de sus ventajas es que permite crear y configurar un diccionario de datos. Un diccionario de datos, a grandes rasgos, permite establecer atributos comunes (por ejemplo, de visualización) que podemos añadir a los nuevos campos que vayamos creando en las tablas.

CONCLUSIÓN

Este mes nos hemos limitado a aclarar algunos conceptos que usaremos a lo largo de la serie. Además, hemos mostrados las herramientas que, más tarde o más temprano, nos veremos obligados a utilizar. El próximo mes abordaremos los componentes que proporciona la librería *VCL* para el acceso y la introducción de datos.

En el próximo número comenzaremos a estudiar los componentes de acceso a datos

Por ahora recomiendo a todos los lectores que se vayan familiarizando con las utilidades (al menos con el *BDE Configurator* y el *DataBase Desktop),* dejando el *SQL* Explorer para más adelante).

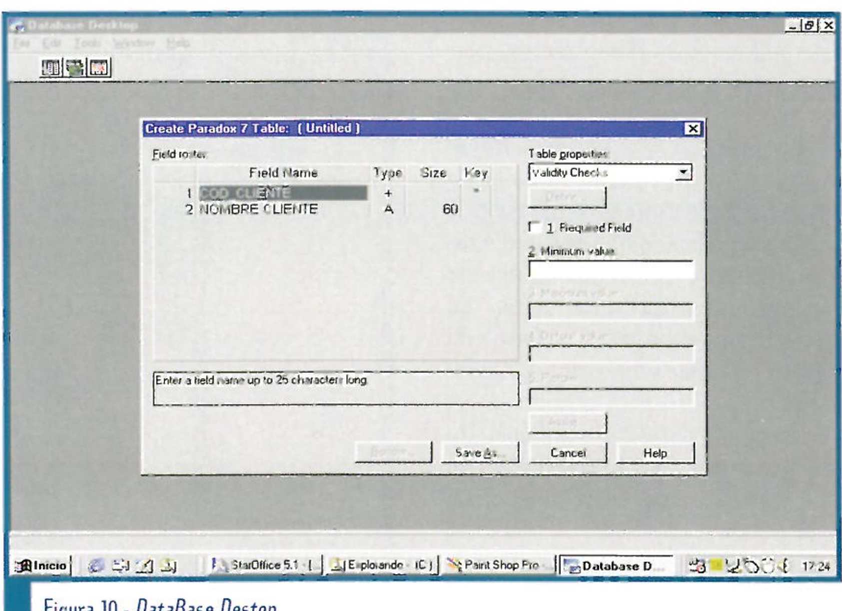
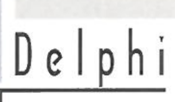
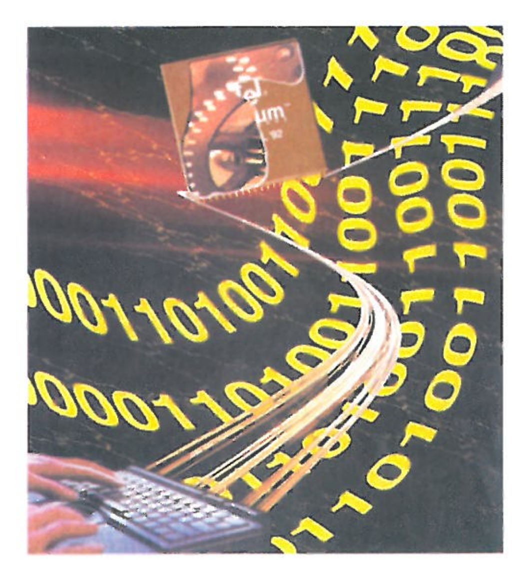


Figura 10.- *DataBase Destop.*





Programación de Threads (y III)

Juan Luis Ceada Ramos.
Programador en ARCABE Formación y Servicios Informáticos.
Octavio Martín Díaz.
Profesor de Análisis de la Universidad de Sevilla.

Para finalizar con la serie, en esta entrega veremos cómo acceder a bases de datos desde múltiples hilos, cómo optimizar la creación de *threads* y la forma de trabajar directamente con la *API* de *Windows*.

OPTIMIZAR LA CREACIÓN DE THREADS

Hay casos en los que una aplicación permanece a la espera de que ocurra un determinado suceso para pasar a la acción. Por ejemplo, supongamos que tenemos un sistema que recibe datos por el puerto paralelo. Al recibir una determinada señal, el sistema lanza un hilo que se encarga de realizar determinadas operaciones. Supongamos que dicha señal se recibe varias veces por segundo.

El proceso es el siguiente: crearemos una instancia de la clase *TThreadList*. En ella almacenaremos las direcciones de memoria de cada uno de los *threads* que vayamos creando. Cuando un determinado *thread* finaliza su ejecución, en vez de ser destruido, simplemente se añade a sí mismo a la *caché*.

Implementar una caché de threads mejora el rendimiento

En estos casos, el proceso de creación/destrucción del *thread* consume un tiempo de *CPU* que puede influir en el rendimiento del sistema. Por conseguir una optimización del sistema es posible mantener una *caché* de *threads*.

Cuando se necesita usar un *thread*, se mira si está en la *caché*. En caso afirmativo se devuelve la dirección de memoria donde comienza. Si no está en la *caché*, se crea uno nuevo. A continuación veremos un ejemplo basado en lo comentado anteriormente, cuyo código se muestra en el Listado 1.

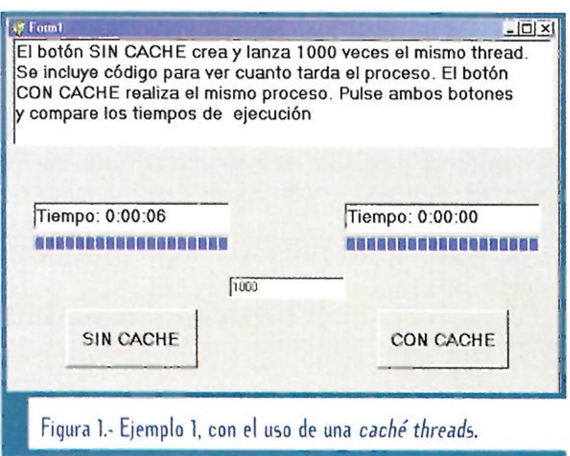


Figura 1.- Ejemplo 1, con el uso de una *caché threads*.



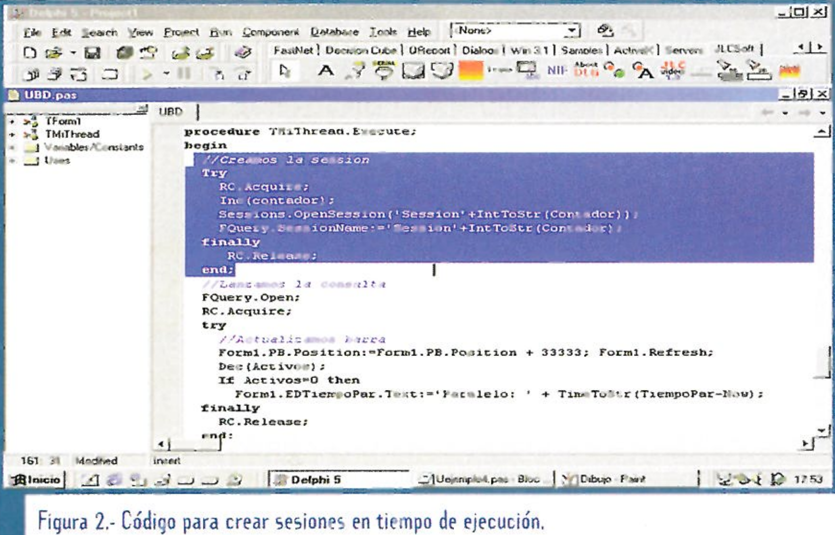


Figura 2.- Código para crear sesiones en tiempo de ejecución.

Se trata de una posible aplicación que lanza el mismo *thread* muchas veces por segundo (como respuesta a una determinada señal), y en la que no interesa perder tiempo en la creación/destrucción del *thread*.

Como se puede observar en el listado, sólo contemplamos la existencia de un *thread* en la *caché*. Para hacernos una idea de cómo funciona este mecanismo es suficiente. Los métodos *LockList* y *UnLockList* hacen posible que muchas instancias del mismo *thread* puedan entrar y salir de la *caché*, sin que haya problemas de concurrencia.

El uso de la caché implica que los threads deben ser liberados manualmente

Hay que tener la precaución de liberar todos los *threads* que existan en la *caché* antes de cerrar la aplicación principal, puesto que éstos no se destruyen por sí solos. Para ello, podemos usar el evento *OnClose* (como se aprecia en el Listado 1). En el *CD-ROM* de la revista se incluye un ejemplo sobre *threads* y *cachés*.

THREADS Y BASES DE DATOS

Si en nuestra aplicación se usan múltiples *threads* para realizar operaciones con bases de datos, tendremos que añadir una nueva sesión por cada *thread*. Tenemos varias opciones: la primera es insertar un componente *TSession* por cada *thread* que vayamos a lanzar en algún formulario.

Puesto que inicialmente es difícil saber cuántos *threads* llegaremos a ejecutar, la mayoría de las veces usaremos la variable *Sessions* para crear nuevas sesiones de forma dinámica. Para ello, llamaremos al método *Sessions.OpenSession*.

Sessions es una variable de tipo *TSessionList* que es automáticamente "instanciada" por las aplicaciones que trabajan con bases de datos.

OpenSession requiere como parámetro el nombre de la sesión a crear, que debe ser única en toda la aplicación. En la Figura 2 se muestra un procedimiento que permite crear sesiones con nombres diferentes en tiempo de ejecución. Usamos una sección crítica para acceder al contador, para evitar los posibles problemas derivados del acceso concurrente a dicha variable.

Puesto que la variable contador es de tipo *integer*, teóricamente podríamos tener miles de millones de sesiones abiertas. En nuestro

Form1

RELLENAR TABLAS | LANZAR CONSULTA SERIE | LANZAR CONSULTA PARALELO

ID_CLIENTE	CLIENTE_NOMBRE
1	NOMBRE CLI 1
2	NOMBRE CLI 2
3	NOMBRE CLI 3
4	NOMBRE CLI 4

ID_AP	APELLIDOS
1	APELLIDOS CLIENTE 1
2	APELLIDOS CLIENTE 2
3	APELLIDOS CLIENTE 3
4	APELLIDOS CLIENTE 4

ID_CLIENTE	CLIENTE_NOMBRE	APELLIDOS
1	NOMBRE CLI 1	APELLIDOS CLIENTE 1
2	NOMBRE CLI 2	APELLIDOS CLIENTE 2
3	NOMBRE CLI 3	APELLIDOS CLIENTE 3
4	NOMBRE CLI 4	APELLIDOS CLIENTE 4

Serie: 0:00:06 | Paralelo: 0:00:06

Figura 3.- Ejemplo 2, acceso a bases de datos desde *threads*.

LISTADO 1. Ejemplos de uso de caché de threads.

```
var
  Cache: TThreadList;

procedure TMiApp.BuclePrincipal;
var MiTh:MiThread;
begin
   while not FinProceso do
   begin
      If RecSenialPuerto then
  begin
            MiTh:=DevolverThread;
           //cambiar prioridad
  //otras operaciones...
   MiTh.Resume;
  //esperamos a que termine
  MiTh.WaitFor;
     end;
   end;
end;

function TMiApp.DevolverThread:TThread;
begin
  with Cache.LockList do
  begin
    //LockList de tipo TList
    if Count=1 then
    begin
      Result := TThread(Items[0]);
      //Sacamos el thread de la
      //cache
      Delete(0);
    end
    else
      //Creamos en modo suspendido
      Result := TMiThread.Create(true);
  end;
  Cache.UnlockList;
end;

procedure TMiThread.Execute;
begin
  //hacer cosas
  //Añadimos el thread a la cache
  Cache.Add(self);
end;

//OnClose de la aplicación
procedure TMiApp.FormClose(Sender: TObject);
begin
  with Cache.LockList do
  begin
    TThread(Items[0]).Free;
    Delete[0];
  end;
  Cache.UnlockList;
end;
```

caso, el contador se inicia a 0 cuando arranca la aplicación y se incrementa indefinidamente.

Form1

Crear un thread en modo suspendido

Reanudar la ejecución del thread

Suspender la ejecución del thread

Finalizar el thread

2434

Figura 5.- Ejemplo 4, manipulación de *threads* mediante la *API*.

En la Tabla 1 se muestran las propiedades y métodos de la clase *TSessionList*.

En el *CD-ROM* se incluye un ejemplo de aplicación que accede a diferentes tablas en paralelo (ver Figura 3). Simplemente, tenemos dos tablas (con 100.000 registros cada una), que han sido rellenadas mediante un simple bucle. La aplicación tiene tres *dbgrids*. En el primero se muestran todos los datos de la Tabla 1 (nombre de clientes). En el segundo, los de la Tabla 2 (apellidos de clientes). El tercero, los datos de una consulta que combina las Tablas 1 y 2. Este último es el que puede causar más problemas, ya que accede a las dos tablas a la vez.

Para implementar la caché, usaremos la clase TThreadList

Antes de lanzar el proceso habrá que rellenar las tablas con datos (botón rellenar tablas). Al tener muchos registros, el espacio en las tablas puede aumentar hasta los 40 *Mb*. Como se aprecia en la Figura 3, no hay excesivas diferen-



TABLA 1. Clase TsessionList.

Nombre	Tipo	Descripción
Count: Integer	Pro.	Número de sesiones creadas.
CurrentSession: Tsession	Pro.	Devuelve la sesión que está activa actualmente.
List: Tsession	Pro.	Devuelve un puntero a la sesión indicada. Ejemplo: List['MiSesion']
Sessions: Tsession	Pro.	Permite acceder a cada una de las sesiones. Ejemplo: Sessions[0]
FindSession	Met.	Busca la sesión que se le pasa como parámetro en la lista de sesiones.
GetSessionNames	Met.	Devuelve una lista (TList) con los nombres de todas las sesiones creadas.
OpenSession	Met.	Activa una sesión existente, o crea una nueva.

cias en el tiempo de ejecución. Pero en sistemas con tres procesadores, el tiempo se dividiría (teóricamente) entre 3.

Nota: Para probar este ejemplo se debe crear un alias llamado PRUEBAS que apunte al directorio donde residan las tablas.

La verdadera potencia de esta forma de acceder a BBDD se aprecia en sistemas multiprocesador.

Cada *TDBGrid* se rellena con datos provenientes de una consulta *SQL*. En el formulario principal se han incluido tres componentes *TQuery* (uno por cada *thread*). Lo hemos hecho por comodidad, puesto que podríamos haber usado únicamente uno (protegiendo el acceso a él mediante una *RC*).

Esto es importante. Gracias a las sesiones, varios *threads* pueden acceder a bases de datos de forma simultánea (el sistema los trata como si fuesen varios usuarios). Pero si comparten el mismo componente *TQuery*, el acceso al mismo debe protegerse mediante algún mecanismo. Piense en esto: ¿qué pasaría si dos *threads* modificasen la propiedad *SQL* del *TQuery* a la vez?

Una ultima cosa: los componentes *TQuery* que usemos deben crearse (ya sea en tiempo de diseño o de ejecución) fuera del *thread*. Si lo creásemos dentro, serían destruidos al destruirse el *thread*. En nuestro ejemplo, esto supondría que los *dbgrids* mostrarían los resultados, e inmediatamente después se quedarían en blanco.

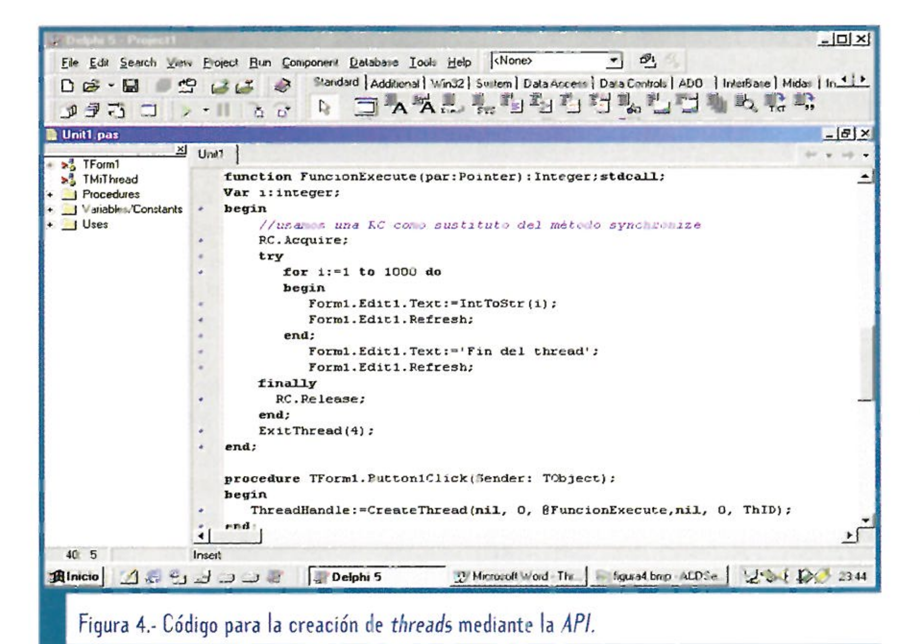


Figura 4.- Código para la creación de *threads* mediante la *API*.

LA API DE WINDOWS

Para crear y lanzar *threads*, *Delphi* proporciona la clase *TThread*. Esta clase encapsula a la mayoría de las funciones relacionadas con hilos de la *API* de *Windows*. De hecho, es muy raro que tengamos que recurrir la *API* para crear aplicaciones multihilo.

Pero hay algunos casos en los que no nos quedará más remedio que usar esta *API*, como puede ser:

- Para crear *threads* con determinadas características relacionadas con la seguridad de los procesos.
- Para sincronizar procesos usando otros mecanismos como semáforos, *mutex*, o variables de interbloqueo (aunque esto, técnicamente, no entra dentro de la creación de hilos).

Veamos ahora cómo crear un *thread* usando la *API* de *Windows*. En la Figura 4 se muestra un código muy simple que crea un *thread*, muestra un mensaje, y finaliza.

Multiple Threads in Delphi - Microsoft Internet Explorer

SBCA Programmers' Pages
An Introduction to Multiple Threads in Delphi

Programming ideas and examples, brought to you by South Bay Computer Assistance.

This tutorial is an introduction to the concept of threads, specifically the thread demo included with Delphi, and isn't meant as an exhaustive treatment of the topic. For best results, print this page and have Delphi open with the thread demo project in front of you.

An application written for 32-bit Windows can consist of more than one process, and a process can consist of more than one thread. The API supports multitasking, which creates the effect of simultaneous execution of multiple processes and threads.

A process is an executing application that consists of a private virtual address space, code, data, and other operating system resources, such as files, pipes, and synchronization objects that are visible to the process. A process also contains one or more threads that run in the context of the process. A thread is the basic entity to which the operating system allocates CPU time. A thread can execute any part of the application's code, including a part currently being executed by another thread. All threads of a process share the virtual address space, global variables, and operating system resources of the process. In a sense, a thread is a program within a program.

A multitasking operating system divides the available CPU time among the threads that need it. In Windows, the Win32 API is designed for preemptive multitasking; this means that the system allocates small slices of CPU time among the competing threads. The currently executing thread is suspended when its time slice elapses, allowing another thread to run. When the

Figura 7.- En *Internet* hay varias páginas donde se puede obtener información relacionada con la programación *multithread*.

En Windows 9X se ignoran los parámetros de seguridad

El primer parámetro de la función *CreateThread* es un puntero a una estructura de tipo *ThreadAttributes*. En ella podremos definir el nivel de seguridad de nuestro *thread*. Lamentablemente, este parámetro sólo es soportado por la *API* de *Windows NT* y no por la de *Windows 95/98*.

En el 99% de los casos no necesitaremos recurrir a la API de Windows

El segundo parámetro indica el tamaño de pila inicial del hilo. Si vale cero, el tamaño de pila será igual al del hilo principal. El tercer parámetro es un puntero a la función que ejecutará el hilo (nuestro método *Execute*). Dicha función debe tener exactamente el formato que se muestra en la Figura 4. Es posible obtener estos fuentes en el *CD-ROM* que acompaña a la revista.

El cuarto parámetro es un puntero a la información que puede recibir la función que pasamos en el parámetro tres. Por último, los parámetros 5 y 6 indican, respectivamente, cómo crear el hilo (suspendido o no), y la variable que almacenará el identificador del hilo (único en toda la aplicación).

Exceptuando el hecho de crear *threads* con diferentes atributos de seguridad (cosa que en *Windows* 9X simplemente es ignorado), no considero que sea necesario recurrir al uso de la *API*, por lo que no profundizaremos más en el tema. En el *CD-ROM* se incluyen algunos ejemplos sobre esto, donde los lectores interesados podrán obtener más información sobre la programación multihilo basada en la *API*. Y por supuesto, no olvidéis consultar el fichero *Win32.hlp*.

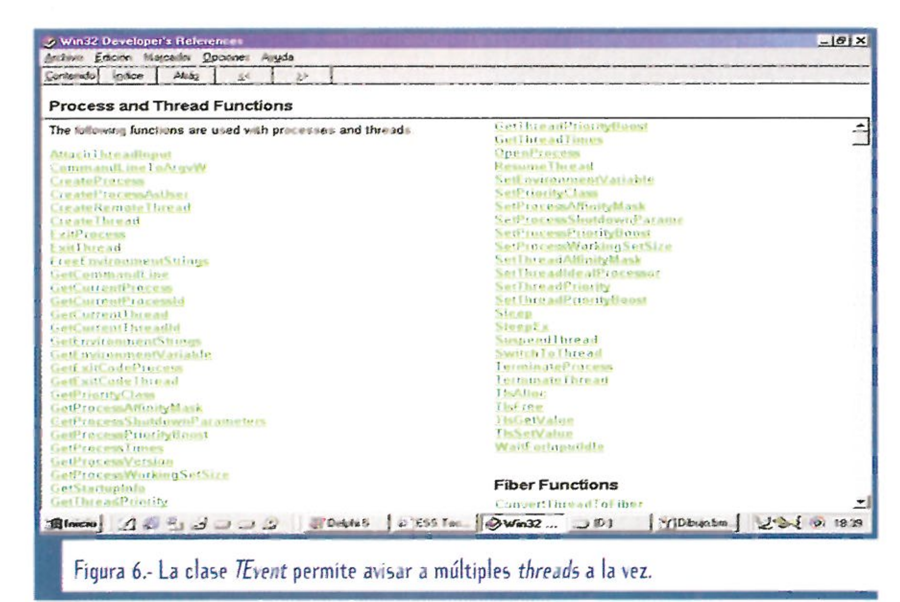
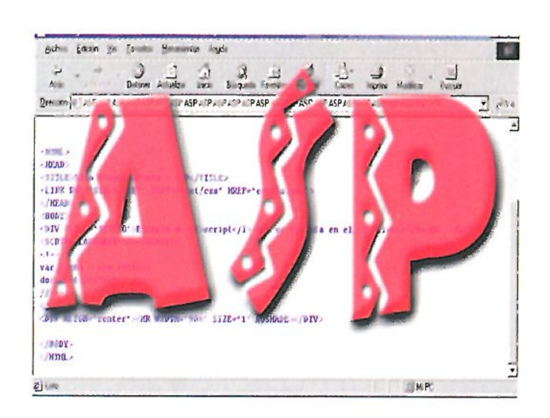


Figura 6.- La clase *TEvent* permite avisar a múltiples *threads* a la vez.





La tecnología ASP (V): ASP y XML trabajando juntos

Adolfo Aladro García. *Analista Programador*

XML ha irrumpido con fuerza en *Internet,* por el momento el único navegador que soporta el estándar es *Internet Explorer 5.0*. Sin embargo, *XML* puede ser utilizado en el lado del servidor, dentro de las páginas *ASP.* Esto nos permite aprovechar las ventajas de *XML* con independencia del navegador.

La tecnología *XML* surge como consecuencia de las limitaciones propias del lenguaje *HTML* en lo que se refiere al tratamiento de los datos. Las páginas *HTML* constituyen una herramienta de diseño y maquetación, pero no son útiles a la hora de manejar datos. Esto es un inconveniente considerable si tenemos en cuenta que los sitios *Web* son cada día más complejos y utilizan diversas fuentes de información con las que se construye dinámicamente las páginas.

EL LENGUAJE XML

Para alguien familiarizado con *HTML* el lenguaje *XML* no presenta ninguna dificultad. Las etiquetas *XML* se diferencian de las etiquetas *HTML* en que las segundas sirven para presentar mientras que las primeras se utilizan para describir los datos. Además, las etiquetas *XML* son creadas por nosotros mismos, siendo diferentes y adaptándose a las necesidades concretas de la información que se desea guardar. Así por ejemplo, si quisiéramos representar una agenda de direcciones de correo electrónico el documento *XML* correspondiente podría quedar como sigue:

```
<?xml version="1.0"?>
<agenda>
  <persona>
    <nombre>Adolfo Aladro
    García</nombre>
    <email>aaladro@arra-
    kis.es</nombre>
  </persona>
  <persona>
    <nombre>Madonna Ciccone</nom-
    bre>

    <email>madonna@ciccone.com</n
    ombre>
  </persona>
  ...
</agenda>
```

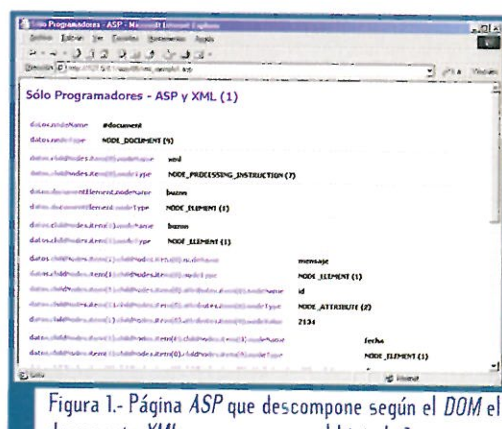


Figura 1.- Página *ASP* que descompone según el *DOM* el documento *XML* que aparece en el Listado 1.



El resultado es un documento de texto plano en el que los datos quedan perfectamente organizados en una estructura en forma de árbol. Hay una raíz única a partir de la cual se agrupa el resto de la información. La primera de las líneas sirve para identificar el tipo de archivo y una vez que es detectada por un *parser* de *XML*, éste se encuentra en disposición de pasar a analizar los datos.

El lenguaje *XML* se caracteriza, tal y como podemos observar, por su sencillez y su capacidad para adaptarse a cualquier fuente de datos.

DOCUMENTOS XML BIEN FORMADOS

Los documentos *XML* bien formados son aquellos que respetan las normas impuestas por el lenguaje *XML*. Siempre que se desarrolle una fuente de datos *XML* hay que observar estas reglas. De otro modo el *parser* de *XML* producirá un error al analizar el documento.

Los lenguajes *XML* y *HTML* guardan una sintaxis similar (de hecho *HTML* es un subconjunto específico de *XML*). La mejor forma de estudiar las reglas que hacen que un documento *XML* esté bien formado consiste en hacerlo desde la perspectiva del lenguaje *HTML*. Sin embargo, esto no debe ser motivo de confusión: estamos hablando en todo momento de *XML*, no de *HTML*.

Figura 2.- *getElementsbyTagName* permite extraer colecciones de nodos donde todos se caracterizan por poseer el mismo nombre.

• Todas las etiquetas deben cerrarse

Esto significa que no pueden darse casos como el que por ejemplo sucede en el lenguaje *HTML* con la etiqueta *<OPTION>*, la cual normalmente no suele ir acompañada de su correspondiente etiqueta de cierre *</OPTION>*; o la etiqueta *
* para la que ni siquiera existe etiqueta de cierre. Las etiquetas que carecen de etiqueta de cierre tienen que llevar el carácter "/" antes del carácter ">" que marca el final. Por ejemplo: *
*.

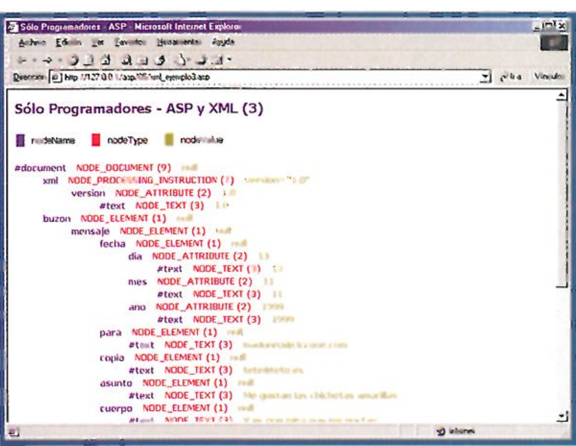


Figura 3.- El *DOM* de *XML* permite recorrer el árbol de datos de forma recursiva.

• El anidamiento de las etiquetas debe ser el correcto

Por ejemplo, no sería válido hacer: *<B><I>···</B></I>*. Lo correcto sería: *<B><I>···</I></B>* ó *<I><B>···</B></I>*.

• Se distingue entre mayúsculas y minúsculas

En *XML* no es lo mismo escribir la etiqueta *<BODY>* que la etiqueta *<body>*, o que la etiqueta *<boDY>*.

• Los atributos deben ir siempre entre comillas

Nunca debe escribirse algo como *<TABLE WIDTH=100 BORDER=0>*, sino *<TABLE WIDTH="100" BORDER="0">*. La primera de las opciones es totalmente válida en una página *HTML*, no así en un documento *XML*.

• Solamente puede haber una etiqueta raíz

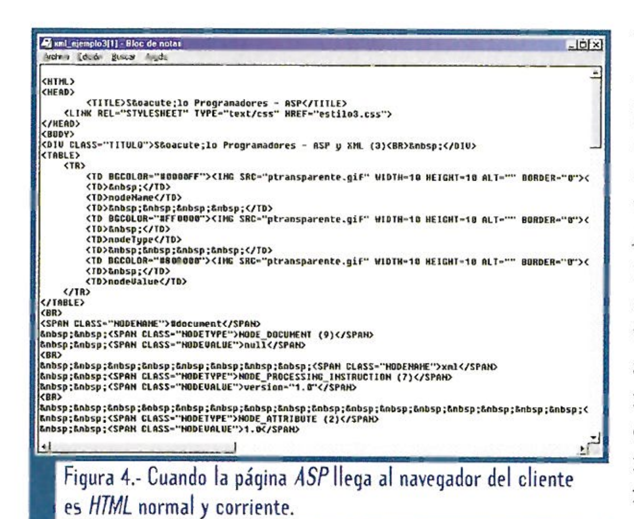
Aplicar esta regla al lenguaje *HTML* sería tanto como decir que las etiquetas *<HTML>* y *</HTML>* tienen que contener a todo el documento.

• Entidades de caracteres

El carácter "<" se escribe dentro de un documento como <. El estándar *XML* solamente define unas pocas entidades de caracteres pero soporta también las entidades numéricas de forma que, por ejemplo, el carácter *"ñ"* se debe escribir dentro de un documento como ñ (Ver msdn.microsoft.com/workshop/author/dhtml/reference/charsets/charsets.asp)

• Bloques de código

Los bloques de código *script* dentro de una página *HTML* pueden contener caracteres que impidan un correcto proceso de *parsing* del documento, por lo que deberán utilizarse mediante secuencias de escape para realizar un documento *HTML* bien formado, o bien el código de *script* se incluirá dentro de una sección



```
<HTML>
<HEAD>
        <TITLE>S&oacute;lo Programadores - ASP</TITLE>
    <LINK REL="STYLESHEET" TYPE="text/css" HREF="estilo3.css">
</HEAD>
<BODY>
<DIV CLASS="TITULO">S&oacute;lo Programadores - ASP y XML (3)<BR> </DIV>
<TABLE>
    <TR>
        <TD BGCOLOR="#0000FF"><IMG SRC="ptransparente.gif" WIDTH=10 HEIGHT=10 ALT="" BORDER="0"><
        <TD> </TD>
        <TD>nodeName</TD>
        <TD>    </TD>
        <TD BGCOLOR="#FF0000"><IMG SRC="ptransparente.gif" WIDTH=10 HEIGHT=10 ALT="" BORDER="0"><
        <TD> </TD>
        <TD>nodeType</TD>
        <TD>    </TD>
        <TD BGCOLOR="#800000"><IMG SRC="ptransparente.gif" WIDTH=10 HEIGHT=10 ALT="" BORDER="0"><
        <TD> </TD>
        <TD>nodeValue</TD>
    </TR>
</TABLE>
<BR>
<SPAN CLASS="NODENAME">#document</SPAN>
  <SPAN CLASS="NODETYPE">NODE_DOCUMENT (9)</SPAN>
  <SPAN CLASS="NODEVALUE">null</SPAN>
<BR>
        <SPAN CLASS="NODENAME">xml</SPAN>
  <SPAN CLASS="NODETYPE">NODE_PROCESSING_INSTRUCTION (7)</SPAN>
  <SPAN CLASS="NODEVALUE">version="1.0"</SPAN>
<BR>
                <
  <SPAN CLASS="NODETYPE">NODE_ATTRIBUTE (2)</SPAN>
  <SPAN CLASS="NODEVALUE">1.0</SPAN>
```

Figura 4.- Cuando la página *ASP* llega al navegador del cliente es *HTML* normal y corriente.

CDATA. Por ejemplo, el siguiente bloque de código:

```
<SCRIPT>
  alert("Hola");
</SCRIPT>
```

debería escribirse como:

```
<SCRIPT><![CDATA[
  alert("Hola");
]]></SCRIPT>
```

Si nuestro documento *XML* cumple las siete normas anteriores el *parser* de *XML* no tendrá ningún problema para analizarlo.

En el Listado 1 del *CD-ROM* se observa un documento *XML* que vamos a utilizar en nuestros ejemplos.

XML distingue entre mayúsculas y minúsculas

La primera de las líneas es una instrucción cuyo propósito consiste en comunicar al *parser* que lo que viene a continuación es un documento *XML*. A continuación aparecen los datos. Como se puede observar se respeta la regla quinta: solamente puede existir una etiqueta raíz.

Un buzón de correo electrónico se compone de mensajes y esto es lo que trata de reflejar la estructura del documento *XML*. Los atributos de las etiquetas están entre comillas, todas las etiquetas se abren y se cierran, se respeta la distinción entre mayúsculas y minúsculas. En resumen, podemos afirmar que el archivo **buzon.xml** contiene un documento *XML* bien formado por lo que el *parser* lo analizará correctamente sin problemas.

EL DOM DE XML

El *DOM* (*Document Object Model*) de *XML* resume la forma en la que podemos acceder a los datos contenidos en un documento fuente. Desde el mismo momento en el que se crea una instancia del objeto que representa la fuente de datos *XML*, se pueden acceder a todas las propiedades y métodos disponibles. Estudiar el *DOM* de *XML* es algo que se extiende mucho más allá de este artículo. Sin embargo vamos a resumir las partes más notables del mismo.

Los datos contenidos en un documento *XML* se estructuran jerárquicamente en forma de árbol. Visto de otra forma podríamos decir que existen listas de nodos, cada uno de los cuales puede apuntar a otra lista de nodos y así sucesivamente. Con *JScript* todas estas colecciones de elementos se suelen manejar de la misma forma.

XML sirve para describir los datos y HTML para presentarlos

EL DOCUMENTO XML

Todo aquello aparece dentro de un documento *XML* que está contemplado dentro del árbol de datos. Esto incluye la sentencia *<?xml version="1.0"?>* y otras como comentarios, etc. Es posible acceder directamente al nodo raíz de los datos utilizando la propiedad *documentElement*. Sin embargo esto no es necesario. El árbol de datos puede tratarse de manera global o analizarse incluso dinámicamente en función de los tipos de nodos que se vayan encontrando.

LOS NODOS DE UN DOCUMENTO XML

Un nodo puede ser un atributo de una etiqueta, una etiqueta, un comentario, etc. En definitiva todos los elementos que aparecen dentro

```
<%@ LANGUAGE = JScript %>
<%

    Function RecorrerArbol(nodo, nivel) {
        var i;

        Response.Write(Blancos(nivel));
        Response.Write('<SPAN CLASS="NODENAME">' + nodo.nodeName + '</SPAN>\n');
        Response.Write("  ");
        Response.Write('<SPAN CLASS="NODETYPE">' + nodeTypes[nodo.nodeType-1] + ' (' + nodo.node
        Response.Write("  ");
        Response.Write('<SPAN CLASS="NODEVALUE">' + nodo.nodeValue + '</SPAN>\n');
        Response.Write("<BR>\n");
        if (nodo.hasChildNodes()) {
            for(i=0;i<nodo.childNodes.length;i++) {
                RecorrerArbol(nodo.childNodes.item(i), nivel+1);
            }
        } else if (nodo.attributes) {
            for(i=0;i<nodo.attributes.length;i++) {
                RecorrerArbol(nodo.attributes.item(i), nivel+1);
            }
        }
    }

    function Blancos(n) {
        var i;
        var resultado = "";

        for(i=0;i<8*n;i++) {
            resultado += " ";
        }
        return resultado;
    }

    // Tipos de nodos
```

Figura 5.- Código fuente de la página *xml_ejemplo3.asp* que muestra cómo se puede recorrer recursivamente el árbol de datos.



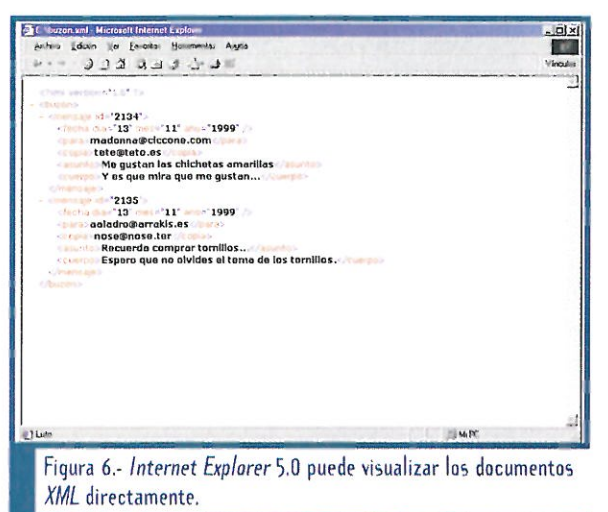


Figura 6.- *Internet Explorer* 5.0 puede visualizar los documentos *XML* directamente.

de un documento *XML* son tratados por el *DOM* como nodos.

Todo lo que aparece en un documento XML se transforma en nodos

- *NodeType:* Valor numérico que representa el tipo del nodo, es decir, si se trata de un elemento, un atributo, etc. La Tabla 1 muestra un resumen de algunos de estos valores

NOTA: existen otros valores, pero para el propósito de este artículo basta con tener en cuenta éstos. Si se desea una información más completa os remitimos a la serie sobre XML que se publicó en estas mismas páginas unos meses atrás.

- *NodeName:* Nombre del elemento. Su valor depende en general del tipo de nodo pero para los nodos que representan elementos y atributos siempre será el nombre de la etiqueta y el del atributo respectivamente.
- *NodeValue:* Normalmente será el valor asociado a un atributo.
- *FirstChild, LastChild* y *ChildNodes:* Primer y último nodo hijo.
- *ChildNodes:* Lista con todos los nodos hijo.
- *NextSibling* y *PreviousSibling:* Nodo hermano anterior y siguiente dentro de la lista de nodos.
- *ParentNode:* Nodo padre.
- *OwnerDocument:* Nodo raíz que contiene al nodo.
- *Attributes:* Lista de atributos asociada a la etiqueta representada por el nodo.

Un documento XML bien formado es el que cumple la sintaxis del lenguaje XML

En cuanto a los métodos cabe destacar *hasChildNodes* que permite conocer si un nodo tiene hijos o no.

LAS COLECCIONES DE ELEMENTOS

Todas las colecciones de elementos que aparecen en el *DOM* presentan la misma estructura y por lo tanto la sintaxis necesaria para recorrerlas es la misma.

El método *item* de una colección recibe como parámetro un valor numérico que indica la posición del elemento de la colección al que se quiere acceder. Devuelve como resultado ese elemento. La propiedad *length* indica el número de elementos que conforman la colección. Estos elementos se numeran del 0 a *length-1*.

PRIMER CONTACTO CON EL DOM

El primero de los ejemplos que vamos a realizar, la página **xml_ejemplo1.asp** utiliza el *DOM* de forma muy simple. Sencillamente carga la fuente de datos y muestra referencias a los nodos que constituyen las ramas del árbol de datos.

En primer lugar debemos cargar el archivo *XML*. Esto se realiza en dos pasos dentro del código *script* de la página *ASP*. El primero de estos pasos consiste en localizar el archivo dentro del servidor:

```
var fichero_datos =
    Server.MapPath("buzon.xml");
```

El objeto *Server* proporcionado por el entorno *ASP* cuenta con el



Figura 7.- *XML.com* es un sitio *Web* donde podemos encontrar abundante información en torno a *XML*.

método *MapPath* el cual establece una correspondencia entre los directorios virtuales del servidor y los directorios reales.

A continuación se procede a cargar el archivo *XML*:

```
var datos =
    Server.CreateObject("Microsof
    t.XMLDOM");
datos.async = false;
datos.load(fichero_datos);
```

A partir de este momento la variable **datos** contiene una referencia a todo el árbol de datos que representa al documento *XML* según el *DOM*. Ahora podemos pasar a experimentar con llamadas al *DOM* con el objeto de familiarizarnos con la estructura que presentan los datos.

El método getElementsByTagName permite obtener listas de nodos con el mismo nombre

La Figura 1 muestra el resultado de visualizar la página **xml_ejemplo1.asp**. En ella se puede observar que, efectivamente, la expresión *datos. documentElement* hace referencia al nodo raíz de datos, y cómo a partir de la variable **datos** se pueden conocer todos los nodos.

También es posible observar que hay expresiones que son equivalentes, como por ejemplo:

```
datos.documentElement.nodeName
datos.childNodes.item(0).nodeName
```

El modelo propuesto por el *DOM* es lo suficiente flexible y versátil como para que se pueda recorrer el árbol de datos de varias formas. De ahí que sea muy fácil encontrar ejemplos como el que acabamos de ver en el que existen expresiones distintas que se refieren a lo mismo.

Finalmente para conocer el nombre de los tipos de los nodos se ha utilizado un *array* auxiliar de datos. La propiedad *nodeType* devuelve un valor numérico y lo convertimos a su nombre accediendo a la posición correspondiente del *array*, como por ejemplo:

```
nodeTypes[datos.nodeType - 1]
```

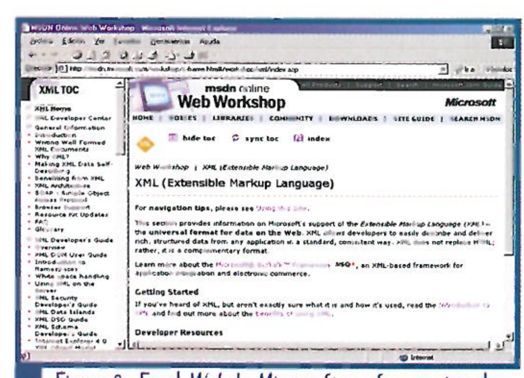


Figura 8.- En el *Web* de *Microsoft* se ofrecen ejemplos donde se combinan *ASP* y *XML*.

ACCEDER A LOS NODOS POR SU NOMBRE

Otra forma de acceder a los nodos del árbol surgido del documento *XML* es el que se presenta en el ejemplo del archivo **xml_ejemplo2.asp**. El método *getElements ByTagName* devuelve una colección con todos los nodos del árbol cuya propiedad *nodeName* coincide con la cadena de texto que recibe como parámetro.

```
var asuntos =
    datos.documentElement.
    getElementsByTagName("asun-
    to");
```

Tras ejecutar la sentencia anterior la variable **asuntos** contiene una lista de todos los nodos **asunto**. Tal y como hemos visto, esa lista puede recorrerse utilizando la propiedad *length* y el método *item*.

```
for(i=0;i<asuntos.length;i++) {
...
}
```

Nótese en este y en el ejemplo anterior que los datos que se sitúan entre etiquetas constituyen en sí mismos también nodos del árbol. Por ello, dentro del bucle anterior, para acceder a los asuntos tenemos que utilizar la expresión:

```
asuntos.item(i).childNodes.item(0)
    .nodeValue
```

Lo mismo ocurre con los atributos de las etiquetas. Éstos constituyen en sí mismos nodos del árbol, y los valores que almacenan también. En el ejemplo anterior, donde veremos cómo desplegar todo el árbol, apreciaremos con mucha más claridad todo esto.

RECORRER EL ÁRBOL DE DATOS RECURSIVAMENTE

En el primero de nuestros ejemplos se accedía directamente a cada uno de los nodos que constituyen el árbol de datos asociado al documento *XML*. Evidentemente esto no tiene una aplicación real. Lo interesante es que mediante una función *JScript* recursiva es posible inspeccionar todo el árbol de datos. En el archivo **xml_ejemplo3.asp** tenemos un ejemplo.



La función *RecorrerArbol* recibe como parámetro un nodo del árbol así como un valor numérico que se utilizará para tabular adecuadamente la salida de forma que la visualización sea representativa de la estructura.

En primer lugar lo que se hace es mostrar las propiedades *nodeName*, *nodeType* y *nodeValue* del nodo que se recibe como parámetro. Para ello se utiliza un código de colores y por supuesto el método *Write* del objeto *Response* proporcionado por el entorno *ASP*.

```
Response.Write(Blancos(nivel));
Response.Write('<SPAN CLASS="NODE-
    NAME">'
   + nodo.nodeName +
    '</SPAN>\n');
Response.Write("  ");
```

La función *Blancos* es un procedimiento auxiliar que sirve para generar cadenas de blancos y que el resultado final tenga un aspecto consecuente con la estructura en árbol.

Se pueden aplicar algoritmos recursivos para recorrer el árbol de datos

A continuación se comprueba si ese nodo tiene hijos mediante el método *hasChildNodes*. Si la respuesta es afirmativa *RecorrerArbol* se llama a sí misma de nuevo para cada uno de los hijos. Además se incrementa el valor almacenado en **nivel** para que cuando los hijos sean pintados aparezcan correctamente tabulados.

```
if (nodo.hasChildNodes()) {

      for(i=0;i<nodo.childNodes.len
      gth;i++) {

      RecorrerArbol(nodo.childNodes
      .item(i), nivel+1);
   }
```

Attributes guarda una lista con todos los atributos de una etiqueta

Puede darse el caso de que un nodo no tenga nodos hijos pero sí nodos atributos que dependan de él. Estos nodos, hijos también de alguna forma, se recorren llamándose *RecorrerArbol* a sí misma tal y como se hace en el caso anterior.

```
} else if (nodo.attributes) {

      for(i=0;i<nodo.attributes.len
      gth;i++) {
      RecorrerArbol(nodo.attribu-
      tes.item(i), nivel+1);
   }
...
```

El resultado obtenido es el que se muestra en la Figura 3. Se puede observar fácilmente la estructura interna de los datos *XML* tal y cómo los interpreta el *DOM*.

COMPATIBLE PARA TODOS LOS NAVEGADORES

La Figura 4 muestra el resultado de hacer ver el código fuente de la página cargada por el navegador. Como se puede apreciar se trata simplemente de lenguaje *HTML*. Esta es una de las ventajas principales que proporciona utilizar *XML* en el lado del servidor en páginas *ASP*. El resultado, es decir, la aplicación *Web* desarrollada no depende en absoluto del navegador.

Es cierto que todos los ejemplos que hemos visto a lo largo de este artículo podrían haber sido ejecutados directamente por el navegador *Internet Explorer* 5.0 dado que soporta *XML*. Pero esta solución por el momento limitaría la audiencia de nuestro sitio *Web*, dejando al margen otras opciones. De ahí que la integración entre *XML* y *ASP* sea, hoy por hoy, una de las alternativas más atractivas e interesantes para empezar a utilizar *XML* en las aplicaciones *Web*.

En el próximo artículo integraremos además las hojas *XSL*, que junto a *XML* y las páginas *ASP*, nos abrirán un mundo de posibilidades muy interesantes.

TABLA 1. Resumen de los valores posibles que puede tomar la propiedad nodeType.

Valor	Significado
NODE_ELEMENT (1)	Elemento.
NODE_ATTRIBUTE (2)	Atributo de un elemento .
NODE_TEXT (3)	Texto contenido en una etiqueta.
NODE_PROCESSING_INSTRUCTION (7)	Instrucción.
NODE_COMMENT (8)	Comentario.
NODE_DOCUMENT (9)	El documento.

Criptografía (y V)

Javier Sanz Alamillo. *Ingeniero de Software.*

En este último artículo se completará la descripción de los algoritmos de clave pública con la presentación del algoritmo *ElGamal.* Además, se mostrarán los conceptos de funciones unidireccionales y firma digital, que actualmente son necesarios para realizar la mayoría de las operaciones de comercio electrónico.

ALGORITMO ELGAMAL

Se trata de un algoritmo muy utilizado y ampliamente implementado, ya que en su momento se presentó como una gran alternativa al *RSA*. Este algoritmo puede ser utilizado tanto para firma digital como para cifrado. Su seguridad se basa en la dificultad del cálculo de logaritmos en campos finitos.

Para generar un par de claves, se elige un número primo *p*, y dos números aleatorios *g* y *x*, tal que *g* y *x* son menores que *p*. Entonces se calcula:

```
y=gx mod p
```

La clave pública está formada por *y*, *g* y *p*. Tanto *g* como *p* pueden ser distribuidas. La clave privada es simplemente *x*. Una vez obtenidas las claves, se realizan los siguientes pasos para poder cifrar/descifrar mensajes con este algoritmo.

Para cifrar un mensaje *M* se genera un número aleatorio *k*, que sea primo a *p-1*. Entonces se calcula:

```
a=gk mod p
b=ykM mod p
```

El par formado por *a* y *b* es el texto cifrado. Para sorpresa de muchos, el tamaño del texto cifrado es el doble que el del texto en claro en cuestión. Para descifrar *a* y *b* se realiza la siguiente operación:

```
M = b/ax mod p
```

Veamos un ejemplo práctico. En el Listado 1, incluido en el *CD*, se observa la utilización de *ElGamal* para cifrar y descifrar un texto de un determinado tamaño.

Para ello, se genera un objeto del tipo **ElGamalKey eg** determinando el tamaño de la clave que utilizar. Esto se realiza en la siguiente sentencia:

```
ElGamalKey eg = new ElGamalKey
    (512 );
```

A continuación se genera la clave y mediante **kp**, **egpub** y **egpri** se pueden utilizar tanto la privada como la pública.

```
kp = ElGamalKey.createKeys(512);
CipherKey egpub = (CipherKey)
    kp.getPublic();
CipherKey egpri = (CipherKey)
    kp.getPrivate();
```

Seguidamente se invoca mediante un *array* de *bytes* con el texto en claro a la función *encrypt()* con la clave pública generada, obteniéndose así el texto cifrado:

```
egpub.encrypt ( claro, 0, cifra-
    do, 0 );
```

Para realizar el descifrado se realiza la llamada a *decrypt()* utilizando **egpri**. En la siguiente línea se muestra este proceso.

```
egpri.decrypt ( cifrado,0,
    claro,0 );
```

También se puede utilizar *ElGamal* para firma digital. Así, para fir-



mar un mensaje M se elige un número aleatorio k tal que sea primo con p-1 y se calculan dos factores que denominaremos a y b. Para el primero se realiza:

```
a= gk mod p
```

y aplicando el algoritmo de *Euclides* para encontrar b, se resuelve la siguiente ecuación:

```
M = (x.a + x.b) mod (p-1)
```

La firma es el par a y b. El valor aleatorio k debe ser guardado en secreto y para verificar una firma, se debe cumplir que:

```
ya ab  mod p = gM mod p
```

En la Figura 1 puede apreciarse la ejecución del Listado 1, con lo que se demuestra el uso de *ElGamal* para el cifrado.

EL MEJOR ALGORITMO DE CLAVE PÚBLICA

Al igual que ocurría con los algoritmos de clave privada, en función de las necesidades del desarrollador un algoritmo de clave pública será más práctico que otro según la situación.

Por regla general, la aceptación más común para la mayoría de las aplicaciones es utilizar *RSA*, pero debido a sus patentes y licencias, más de uno prestará atención antes a los costes que a los beneficios de su uso en el desarrollo.

Aunque el *RSA* puede utilizarse tanto para cifrar como para firmar, siempre se puede utilizar *ElGamal* y una fuerte función de resumen (que se detallan a continuación) para obtener resultados similares y muchas veces mejores.

Para el intercambio de claves, actualmente *Diffie-Hellman* sigue siendo la mejor opción, por su sencillez y carencia de patente.

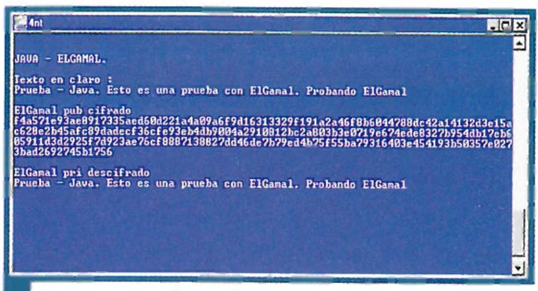


Figura 1.- Ejemplo de *ElGamal* como algoritmo de cifrado.

Recuerde, antes de elegir un algoritmo, con independencia del tipo, consulte la bibliografía presente para asegurarse de que no haya sufrido ningún ataque con éxito.

FIRMA DIGITAL Y AUTENTICACIÓN

Un concepto importante en la seguridad de la información es el de *firma digital*. Como se ha podido observar en los anteriores algoritmos, todos ellos tienen una función específica y en determinados aspectos dejan abiertas algunas incógnitas.

La mayoría de los criptosistemas de clave pública mantienen la *autenticidad* del usuario mediante su clave privada, ya que se da por hecho que no hay desconfianza entre los sujetos que están intercambiando información.

Pero si se considera a un sujeto A como emisor y a otro B como receptor, nadie garantiza a B que el mensaje que recibe de parte de A provenga de quien dice. No hay garantía de que ese mensaje fuera creado y enviado por A. ¿Quién me garantiza la autenticidad?. Todo el mundo conoce la clave pública de B.

Del mismo modo que ocurre esto, tampoco nadie nos asegura de alguna forma si el mensaje recibido de una persona pertenece a la misma, o sea, que éste pueda negar que lo ha enviado ella (no-repudio).

En definitiva y como veremos a continuación, los mecanismos de autenticación garantizan un ataque de terceros en el contenido, pero no proporcionan las características de la firma manual (y para este caso la digital) tan necesarias para muchas operaciones.

PROPIEDADES

Las principales propiedades de la firma manual son las siguientes:

- Fácil y barata de producir.
- Fácil de reconocer.
- Imposible de rechazar por el propietario.
- Infalsificable, al menos en teoría.

De todas estas características cabe destacar que la firma digital, en contra de la manual no tiene porqué ser siempre la misma, puesto que, como es de esperar, sería fácilmente falsificable.

Puesto que la firma digital va a ser una serie de *bits*, se debe for-



Figura 2.- Función de mezcla o resumen.

mular un método para generar la firma digital en cada momento y que identifique al sujeto que la produce, en resumen, que cumpla las anteriores propiedades.

Formalmente, consideremos al conjunto M como el conjunto de todos los posibles mensajes que un sujeto A puede generar y a F como el conjunto de todas las firmas digitales que puede generar. Si proponemos F_A como la función de transformación que utiliza el sujeto A para firmar sus mensajes, existe un m'Œ M / $F_A(m')$ es la firma del mensaje m' por el usuario A.

Esta firma, $f' = F_A(m')$, f'Œ F, se puede añadir al mensaje, por lo que el par mensaje-firma (m',f') se denomina *mensaje firmado*.

En estas circunstancias, el receptor del mensaje puede comprar y verificar la firma, por lo que es posible autenticar el mensaje m'. Puesto que F_A es una función privada para el sujeto A, se necesita de otra transformación para verificar la firma de A.

Se define entonces una función de verificación V_A como la transformación que verifica la firma de A y se basa en devolver un resultado de verdadero o falso de la relación (m',f'). Por tanto, V_A debe cumplir una serie de propiedades:

Que la función V_A devuelva verdadero, V_A (m',f'), si y sólo si $F = F_A(m')$.

No es posible realizar cálculos tal que se pueda obtener m' y f' de V_A (m',f')=verdadero, excepto para el sujeto A usando F_A.

Aunque todo pueda parecer muy enrevesado, de lo que se trata en resumen es de conseguir una función tal que a partir de un texto cualquiera, se genere un resultado que no se pueda falsificar, alterar y demuestre que únicamente una determinada persona lo ha podido generar.

Veamos un sencillo ejemplo de cómo se genera una firma digital usando *RSA*. Presentamos a dos sujetos A y B que utilizan *RSA* y cuyas claves públicas son conocidas.

ElGamal se presenta como una buena alternativa al uso de RSA

Para enfocar el proceso, definimos los pares de claves, para el sujeto A (E_A, D_A) y para el B (E_B, D_B), como las respectivas claves pública y privada. Si el sujeto A quiere enviar un mensaje firmado B tendría que hacer lo siguiente:

1. Se genera un mensaje firmado mediante el cifrado del texto en claro de A usando su clave privada D_A por lo que se obtiene el mensaje firmado, $MF_A = D_A$(M)

2. Ahora el sujeto A cifra el mensaje firmado usando la clave pública de B, E_B, y envía el resultado.

3. El sujeto B ahora procede a descifrar el contenido que ha recibido. Según le dicen, procede de A pero nadie se lo garantiza. Usa su clave privada D_B para descifrar el mensaje y obtiene un mensaje firmado por alguien. Según le dicen, debe ser A, pero nadie se lo garantiza. Como resultado obtiene MF_A.

4. Como debe ser de A, utiliza la clave pública de A, E_A y la aplica sobre el mensaje firmado MF_A. Si el mensaje proviene de A podrá descifrarlo y obtener así el mensaje en claro M, si no, no podrá descifrarlo por mucho que pruebe o intente.

Como se aprecia, de este modo se consiguen resolver algunos problemas planteados.

Por ejemplo, B sabe a ciencia cierta que el mensaje proviene de A, puesto que ha utilizado la clave pública de éste y como nadie puede reproducir la clave privada de A, esto se lo confirma. Se ha podido autenticar al sujeto A.

Por otro lado, B tampoco puede modificar nada de lo que le ha enviado A, ya que tendría que reproducir el mensaje firmado y al no conocer la clave privada de A le es imposible. Como B no puede modificar el contenido y el proceso se ha desarrollado correctamente, A no puede alegar que alguien envió el mensaje por él, ya que nadie excepto él conoce su clave privada (no-rechazo).

Aunque este sistema pueda parecer ideal, en determinados casos prácticos resulta algo incómodo. Por ejemplo, cuando el mensaje que se va a manejar no tenga una longitud fija o no pueda ser tratado como un todo sino que se gestiona en pequeños trozos, con el método que presenta *RSA* tendríamos que firmar cada uno de los trozos, sien-

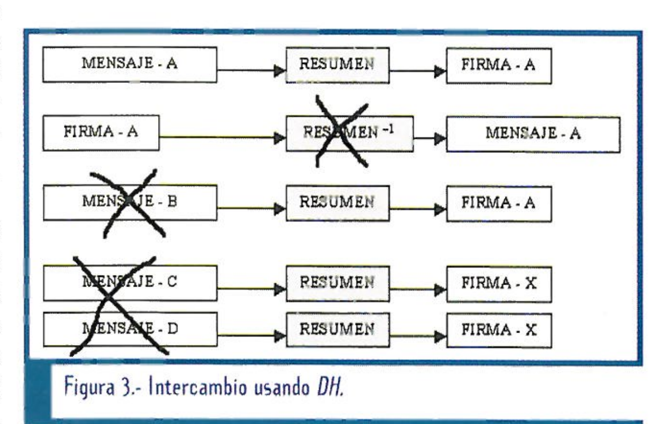


Figura 3.- Intercambio usando *DH*.



do esto un proceso relativamente lento con el inconveniente de que habría que controlar el orden de los fragmentos para obtener la firma final, ¿están todos?, etc.

Para resolver estos problemas se desarrollaron las *funciones de mezcla o resumen (unidireccional)*, también denominadas *message digest* o *fingerprint*.

En la Figura 2 se puede visualizar los elementos para la creación de una firma a partir de una función de resumen. Una función de resumen es aquella que cumple las siguientes propiedades:

- Debe ser simple y rápida de calcular.
- Debe convertir un mensaje de una longitud arbitraria *M* en uno de longitud fija *h(M)* denominado resumen de *M*.
- Debe ser unidireccional, esto es, dado un valor *y* debe ser imposible encontrar un *M* tal que *h(M) = y*.
- Se debe poder asegurar que es imposible encontrar dos mensaje *M* y *M'* tales que *h(m)=h(M')*. Esto se denomina imposibilidad de colisión.

En la Figura 3 el lector puede ver gráficamente el conjunto de propiedades.

Por tanto, podemos utilizar la función resumen sobre todo el texto en claro a enviar. El resultado es un bloque de datos de un tamaño determinado, que cifrado con la clave privada del origen, se envía con el resto de texto en claro que es cifrado a su vez utilizando la clave pública del destinatario. Mediante este sistema, podemos proporcionar las mismas garantías que en el anterior ejemplo.

Existen una serie de funciones de resumen muy conocidas, como son *MD5* y *SHA* que se muestran brevemente a continuación, para seguir con un ejemplo sobre cómo generar firma digital usando *Java*.

La firma digital posibilitará el desarrollo de cualquier transacción económica de forma segura

En la Figura 4 puede observar el proceso que seguiría un mensaje en claro que se firma. Sin incluir la clave pública que utilizar sobre todo el mensaje que se va a enviar, que incluye la firma, se aprecia que a partir de un texto se obtiene, mediante una función de resumen, una firma que es cifrada con la clave privada del emisor.

Todo el mensaje, texto en claro y firma, se cifraría con la clave pública del destinatario. El destinatario al recibir el mensaje utiliza su clave privada, con lo que obtiene el texto en claro y la firma del emisor.

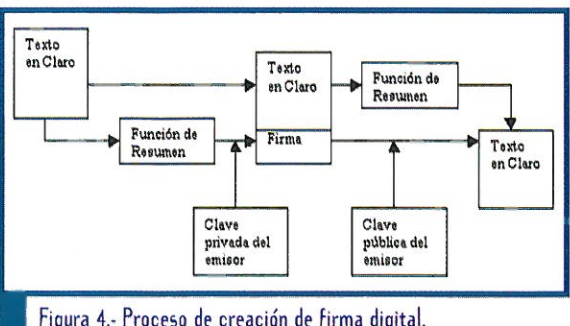


Figura 4.- Proceso de creación de firma digital.

La única forma de averiguar quién ha enviado el texto es usando la clave pública del emisor, con lo que una vez realizado esto se obtiene el resumen. Si al aplicar la función de resumen sobre el texto en claro éste coincide con el conseguido al aplicar la clave publica del emisor, se tienen todas las garantías de que el emisor es quién dice ser y que el texto recibido no ha sido modificado.

ALGORITMO MD5

Se trata de uno de los algoritmos más conocidos, y es el resultado de una mejora sobre *MD4* creado por *Ron Rivest*. Procesa bloques de 512 *bits* y produce una salida de 128 *bits*.

Su funcionamiento es algo complejo, pero se puede resumir en que dado un mensaje, éste se adapta hasta que es 64 *bits* inferior a un múltiplo de 512 *bits*. Se realiza un relleno utilizando un 1 seguido de los 0 que sean necesarios.

A continuación, se añaden 64 *bits* que representan la longitud del mensaje original, sin contar los añadidos. Seguidamente, se inicializan una serie de registros de 32 *bits* y se realizan varias iteraciones con esos bloques de 32 *bits*. Como resultado, tenemos 4 bloques de 32 *bits* que forman el resumen de 128 *bits*.

ALGORITMO SHA

Este algoritmo ha sido desarrollado por el *NSA* y se considera seguro y libre de puertas traseras. Produce firmas de 160 *bits* a partir de bloques de 512 *bits* del mensaje original. El algoritmo es similar al *MD5*, sólo que en vez de utilizar cuatro registros de 32 *bits* utiliza cinco.

Una vez que se han mostrado los conceptos elementales sobre firma digital, se va a desarrollar un ejemplo de cómo generar firma digital utilizando *Java*. En el Listado 2 del *CD-ROM* puede observarse este proceso.

En el método *firmaDigital()* se genera una clave utilizando el algo-

ritmo de *ElGamal*, aunque puede usarse si se prefiere *RSA*. Una vez generadas las dos claves, pública y privada, se asignan a distintos objetos tipo *Key* para su posterior manipulación. Por tanto disponemos de **key_egpri** y **key_egpub**, siendo la clave privada y pública generadas respectivamente.

Una vez que disponemos el texto en un *array* de *bytes*, que se denomina **cifrado**, utilizamos el método *create* de la clase *Fingerprint* para aplicar la función de resumen, eligiendo para ello un algoritmo del tipo *SHA*, específicamente **SHA1**. Con ello, conseguimos un bloque como resultado de una función resumen que se almacenará en un objeto de tipo *Fingerprint*, llamada **fp**.

Realizado esto, se invoca al método *sign* de la clase *SignatureKey* con la clave privada y sobre la función resumen, con lo que obtenemos la firma.

Para comprobar que la firma sea correcta se dispone del método *verify* en la clase *SignatureKey* que utilizado con la clave pública y pasándole la firma y el resultado de la función resumen, devuelve verdadero o falso como verificación, como se detalló anteriormente.

LA MEJOR FUNCIÓN DE RESUMEN

Como ocurre con los algoritmos de clave privada o pública, el uso de una función u otra depende de muchos factores, experiencia, necesidades, requerimientos planteados, patentes, tiempo de ejecución, etc. En la actualidad, la gente se decanta por las dos grandes, *MD5* u *SHA1*.

MD5 es hoy en día muy utilizada y existen muchas implementaciones, pero debido a los rumores de que se ha llegado a demostrar que se puede atacar con cierta "facilidad", por su escaso tamaño, ha dejado paso a *SHA1*, que tiene un mayor tamaño y ha sido diseñada por una organización con mucha experiencia y que sabe lo que hace, esperemos, como es la *NSA* (*National Security Agency*). Sólo el tiempo decidirá cuál de las dos se alza victoriosa o si se seguirán utilizando indistintamente.

OTRAS LIBRERÍAS

Si bien hasta ahora se ha venido utilizando la librería *Cryptonite* para realizar los ejemplos prácticos utilizando *Java*, existen en el mercado una gran variedad de librerías, tanto comerciales como de libre uso, que pueden ser de interés para el lector. Entre las más conocidas cabe destacar:

1. *Java.crypto.toolkit*, de *KonaSoft*, comercial.
2. *JCP* de *Jamie Band*, que incluye *SSL* 3.0, comercial.
3. *IAIK*-JCE, incluye *RSA*, de *Wolfgang Platzer*. Hay versión gratuita y comercial, con especial interés en licencias aparte por el código fuente.
4. *Nate Sammons* ofrece unas clases gratuitas pero únicamente dispone de *DES, TripleDES* y *SHA*.
5. *Acme* de *Jef Poskanzer*. Gratuita.
6. *Pegwit* de *George Barwood*, implementa un paquete de uso con curvas elípticas.
7. *SSLava*, de *Phaos*, implementa *Java SSL*.
8. *Cryptix, de Systemics Ltd.* Gratuita para aplicaciones no comerciales. Muy extendida y de cierta robustez de uso.
9. *Baltimore J/Crypto* ofrece *Jsafe*, comercial.
10. *Java wrapper* de *Ming-Ching*, que ofrece un acceso nativo mediante unas librerías en *C*.
11. *Sun* ofrece *JCA-JCE (Java Cryptography Arquitecture - Extension)* que incluyen entre otros *DES* y *RSA*. Además, permite el tratamiento de certificados estilo *X.509*.

De todas ellas, cabe destacar principalmente dos, que son las que se van a presentar a continuación.

CRYPTIX

Esta librería contiene un amplio conjunto de clases que resuelven las principales necesidades criptográficas más comunes a la hora de desarrollar aplicaciones. Aunque la implementación que nos interesa es la escrita en *Java*, también existe otra paralela en *Perl*.

> Hoy en día se dispone de un amplio conjunto de librerías gratis y muy robustas

Si como ya se ha comentado, por la cantidad de operaciones y cálculos algunos algoritmos son lentos por definición, su imple-



Figura 5.- Ejemplo de firma.



mentación en *Java* no implica que automáticamente ese desarrollo vaya a mejorar en velocidad de ejecución. Puesto que *Java* es, frente a otros lenguajes de ejecución, algo menos rápido, muchas clases tienen una implementación nativa que hace que la librería no sea totalmente portable, aunque sí mucho más rápida de lo que se espera.

Existen versiones de *Cryptix* para *Windows 95/98/NT*, *Solaris* (*SPARC*), *Linux* (*x86*) e *IRIX* (*MIPS*). En la distribución se pueden encontrar tanto los binarios como los fuentes.

Si no encuentra o le venden el código fuente de una librería criptográfica no la compre/pruebe. Nadie le puede garantizar lo que pueda estar realizando internamente esa librería, y más aún si lo que realiza es correcto.

Esta es una de las principales razones por las que casi nadie se atreve a utilizar la *API* de criptografía que ofrece *Windows*. No ofrece el código fuente, al ser muy poco utilizado cabe esperar que tenga algún que otro error "serio" y el emisor de esa *API* no tiene precisamente buenos antecedentes en los entornos de seguridad, en resumen, no es nada recomendable.

Cryptix es gratuita, no hay que pagar *royalties*, salvo para aplicaciones comerciales. Lo curioso del asunto es que los desarrolladores, *Cryptix Development Team* de *Systemics Ltd.* comenzaron publicando el código porque creyeron que no era un negocio rentable.

Mediante el uso de esta librería se dispone de clases para utilizar *DES, TripleDES, IDEA, Blowfish, Speed, Cast5, Loki91, RC2, RC4, RSA, ElGamal, MD5, SHA*, etc.

Puesto que la librería está desarrollada en *Java*, hacen falta unos mínimos requerimientos sobre las máquinas virtuales que utilizar, pero cabe destacar que estando al día en las mismas, por ejemplo, con la descarga gratuita del último *JDK* del *Web* de *Sun* podrá utilizar *Cryptix* sin problemas.



Figura 6.- Librería *Cryptix*.

Entre las características que se están intentando desarrollar está la de ofrecer una funcionalidad estilo *PGP*, cumplir *IJCE* (*International Java Cryptography Extension*), implementar el *SSL* (*Secure Socket Library*) de *Netscape* y mejorar la velocidad de ejecución.

IMPLEMENTACIONES DE SUN PARA JAVA

Sun ha definido *JCA* y *JCE*. *JCA* (*Java Cryptography Arquitecture*) es un *framework* incluido en el *JDK 1.2* que ofrece la posibilidad de utilizar un amplio conjunto de algoritmos para el desarrollo de aplicaciones seguras.

Entre las ventajas que ofrece está la capacidad de utilizar otras librerías como si fuera la propia, lo que *Sun* define con las propiedades de extensibilidad e interoperatividad.

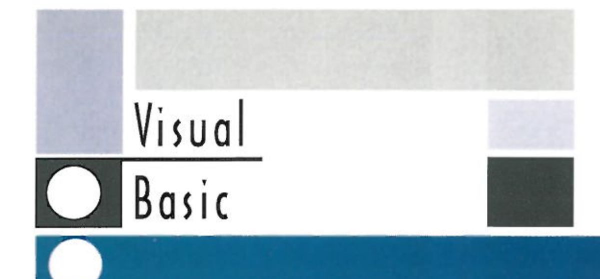
Para realizar determinadas tareas, cifrados e intercambios de claves, es necesario descargar del *Web* de *Sun* el *JCE* (*Java Cryptography Extension)*, que ofrece un conjunto de algoritmos, modos de operación, etc. Se debe recordar que para descargar este extra hay que ser ciudadado estadounidense o canadiense. Debido a las restricciones de seguridad, no es legal descargar este *software* desde otro sitio que no sea norteamérica.

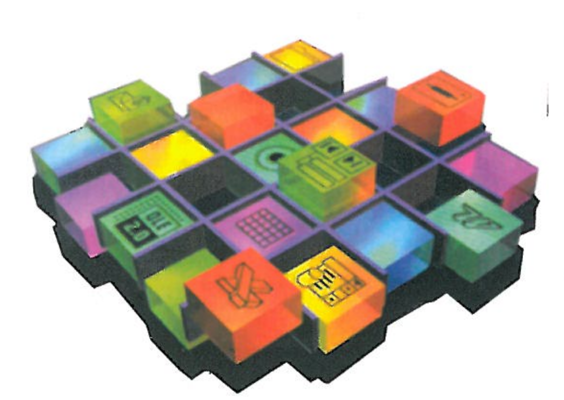
Descargar el *software* implica que se está realizando según la legislación norteamericana de tráfico de armas, puesto que el *software* de seguridad se trata como armamento. No es recomendable jugar con este tema, y si no lo creen, pregunten al creador del producto *PGP*, que pasó cierto tiempo a la sombra hasta que se resolvió su "problema".

Entre los algoritmos disponibles con el *JDK-JCE* de *Sun* destacan *MD5, SHA1, DES, RSA* y *DSA*. Debido a la cantidad de librerías disponibles, actualmente no se conocen aplicaciones comerciales que usen *JCA/JCE*, y por sus restricciones de exportación, es de esperar que no haya sido muy depurada y que no sea muy conocida entre los grandes desarrolladores de aplicaciones sobre seguridad, especialmente los europeos.

CONCLUSIÓN

Con la presentación de *ElGamal* se ha mostrado el conjunto más interesante y utilizado de algoritmos de clave pública. Se explicaron los conceptos relacionados con la firma digital, acompañados de varios ejemplos para que el lector pudiera comprobar su importancia práctica, ya que posibilita el desarrollo de transacciones electrónicas de forma segura.





Suite para Internet (y V): Posibilidades adicionales

Jordi Agost Moré.
Profesor de Programación/Multimedia de la Universidad de Lleida.

¿Cómo podemos arrancar automáticamente una conexión a *Internet*? ¿Cómo podemos utilizar *Outlook* como gestor de correo? ¿De qué forma podemos saber el estado de todos los puertos de comunicación? A estas y otras preguntas intentaremos responder en el último artículo de esta serie.

ARRANCANDO AUTOMÁTICAMENTE UNA CONEXIÓN

En primer lugar vamos a ver una función muy simple, pero no por ello menos interesante. En ella lo que vamos a hacer es realizar de forma automática una conexión a un determinado proveedor de servicios.

Para ello ejecutaremos mediante la orden *Shell* de *Visual Basic* la *DLL rnaui.dll* con el programa *rundll32.exe* y le indicaremos que, de todas las conexiones que tengamos disponibles, ejecute la que deseemos. Para saber las conexiones que tenemos disponibles en nuestro sistema, así como su nombre bastará con ir a *MiPc* dentro del apartado de acceso telefónico a redes y allí encontraremos todas las conexiones con los *ISP* (*Internet Service Provider* : Proveedor de Acceso a *Internet*) de que disponemos junto con el nombre que utilizaremos en la rutina.

En este caso la conexión con el *ISP* utilizada se llama **Xarxa UDL-Net**. Una vez que hemos lanzado la pantalla con nuestra conexión a *Internet*, lo que haremos es enviarle la tecla **Enter**, que es lo que equivaldría a pulsar manualmente el botón de conexión.

Y puesto que la carga y pulsación de botón no son instantáneas insertaremos dos órdenes *DoEvents* para dar así tiempo al sistema a procesarlas. A continuación examinemos la rutina:

```
Private Sub Command1_Click()
  Dim X

  X = Shell("rundll32.exe
    rnaui.dll,RnaDial " &
     "Xarxa UDL-Net", 1)
  DoEvents
  SendKeys "{enter}", True
  DoEvents
End Sub
```

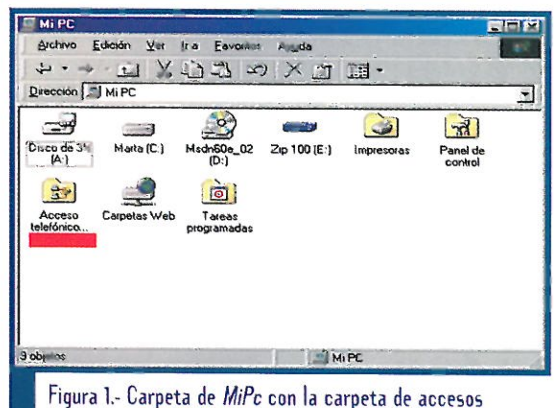


Figura 1.- Carpeta de *MiPc* con la carpeta de accesos telefónicos a redes.



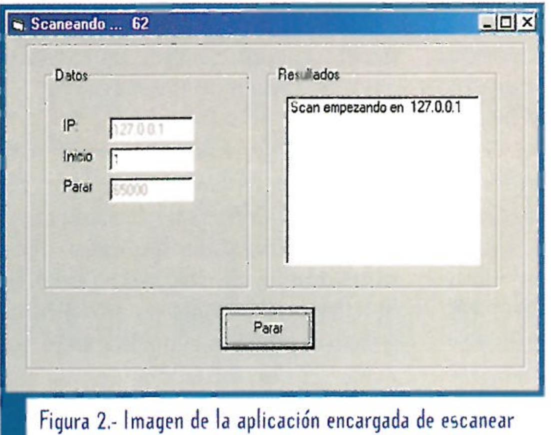


Figura 2.- Imagen de la aplicación encargada de escanear puertos.

ESCANEADOR DE PUERTOS

A veces es muy interesante, sobre todo en el campo de las comunicaciones, conocer el estado de nuestros puertos de comunicaciones; conocer cuáles de ellos están activos y cuáles no. A continuación vamos a realizar un pequeño programa que nos permitirá ver el estado de cada puerto de un equipo (dada su dirección *IP*). Además podremos establecer el puerto de inicio donde queremos que empiece a escanear y el puerto final.

En MiPc encontraremos la carpeta de conexiones telefónicas a redes donde veremos todas las conexiones existentes

Para realizar todo lo explicado anteriormente deberemos añadir a nuestro programa un control *Winsock* que será el encargado de establecer la comunicación con el puerto y un control *Timer* que nos dará la señal para empezar a observar otro puerto.

Asimismo declararemos una variable global al formulario, llamada **booEscaneado**, que indicará en cada momento si el escaneo de puertos se está llevando a cabo o no.

Al iniciarse el programa asignaremos la variable a **False** dado que aún no hemos empezado el proceso:

```
Private Sub Form_Load()
  booEscaneando = False
End Sub
```

Veamos a continuación la rutina que controlará todo el proceso y que es activada por el botón de inicio:

```
Private Sub cmdControl_Click()
  If cmdControl.Caption =
    "Inicio" Then
    Timer1.Enabled = True
    txtInicio.Enabled = False
    txtParar.Enabled = False
    txtIP.Enabled = False
    wskTmp.RemoteHost =
    txtIP.Text
    lst1.AddItem "Escaneado empe-
    zando en  " & txtIP.Text
    cmdControl.Caption = "Parar"
 Else
    Timer1.Enabled =
    False
    txtInicio.Enabled =
    True
    txtParar.Enabled =
    True
    txtIP.Enabled =
    True
    booEscaneando =
    False
    lst1.AddItem
    "Escaneado deteni-
    do en el puerto "
    & lblPort.Caption
    cmdControl.Caption
    = "Inicio"
  End If
End Sub
```

En ella vemos si el botón no ha sido pulsado (es decir en él aparece la palabra **Inicio**), en cuyo caso activaremos el control *Timer*, desactivaremos las etiquetas para que el usuario no pueda modificar los valores previamente introducidos, notificaremos en la lista de mensajes que hemos empezado a hacer el escáner y cambiaremos el mensaje del botón.

Suponiendo que el botón haya sido pulsado entonces haremos la operación contraria: detendremos el escaneo de puertos, se lo comunicaremos al usuario, activaremos las etiquetas por si desea cambiar algún valor y la variable que controla si estamos escaneando o no (**booEscaneando**) la estableceremos a **False**.

Recordemos a su vez que el control *Winsock* posee un evento llamado *Connect* que se ejecuta cada vez que el control logra establecer una conexión.

Es en dicho evento donde le comunicaremos al usuario si la conexión con un determinado puerto ha tenido éxito o no:

```
Private Sub wskTmp_Connect()
  lst1.AddItem "Connected: " &
```

Figura 3.- Activamos la referencia a las librerías de *Microsoft Outlook*.

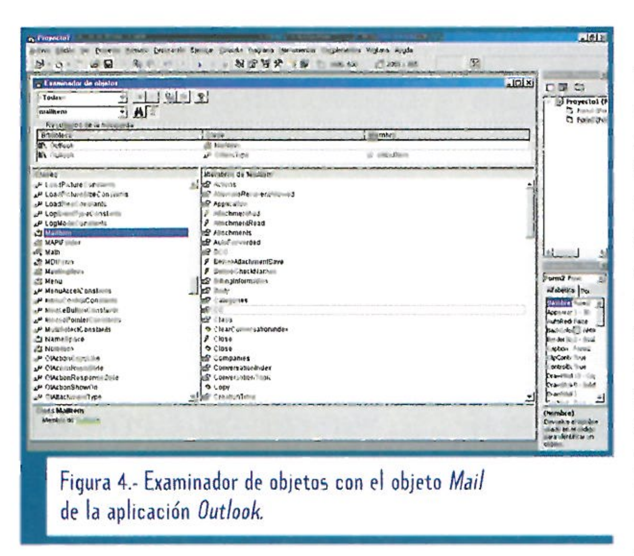


Figura 4.- Examinador de objetos con el objeto *Mail* de la aplicación *Outlook*.

```
    wskTmp.RemotePort
End Sub
```

Llegado a este punto vamos a examinar la rutina que realmente constituye el corazón del sistema y que es en la que se ejecuta el control *Timer* cuando pasa el intervalo de tiempo especificado. En dicha rutina suponiendo que el proceso de escaneado no esté activado, lo que vamos a hacer va a ser establecer la propiedad del puerto remoto del control *Winsock* (propiedad *RemotePort*) al valor que el usuario haya establecido para el puerto inicial.

Tendremos un control Winsock para hacer la comunicación y un control Timer para saber cuándo observar otro puerto

Y en caso contrario, es decir, suponiendo que el proceso de escaneado ya se encuentre en marcha, lo que haríamos sería incrementar el valor del puerto hasta llegar al valor del puerto final con el que se intenta establecer la comunicación.

En ambos casos y una vez establecido el valor del puerto, el control *Winsock* intentará conectarse a dicho puerto y si lo logra mostrará el mensaje tal y como hemos visto anteriormente En caso contrario cerrará la comunicación para que en el próximo lapso de tiempo lo pueda intentar por otro puerto. A continuación veamos la rutina del control *Timer*:

```
Private Sub Timer1_Timer()
    If booEscaneando = False Then
        wskTmp.RemotePort =
      txtInicio.Text
        booEscaneando = True
    Else
        wskTmp.RemotePort =
      wskTmp.RemotePort + 1
    End If
    wskTmp.Connect
    lblPort.Caption =
      wskTmp.RemotePort
    Me.Caption = "Escaneando ...
      " & wskTmp.RemotePort
    wskTmp.Close
End Sub
```

COMPROBAR LA VALIDEZ DE UNA DIRECCIÓN MAIL

A continuación analizaremos un método muy sencillo para saber si una dirección de correo electrónico es válida o no. Este método puede ser útil como validación cuando se pide a un usuario una determinada dirección.

En la siguiente rutina lo primero que miraremos es si se encuentra dentro de la cadena el símbolo de arroba. También comprobamos si el carácter siguiente al carácter arroba es un punto, hecho que indicaría que existe un error en la dirección de correo electrónico dada:

```
Public Function
    ComprobarMail(strMail as
    String) As Boolean
    Dim intl as Integer
    intl = Instr(strMail, "@")

    If intl > 2 Then
        ComprobarMail =
    (InStr(intl + 1, strMail,
    ".") > 0)
    Else
        ComprobarMail = False
         End If
End Sub
```

UTILIZANDO OUTLOOK COMO GESTOR DE CORREO

Aunque en anteriores artículos vimos cómo realizar un cliente de correo *POP3*, a veces no es lo más práctico, sobre todo si nuestra aplicación no realiza un uso intensivo de dicha característica. En estos casos, a veces lo más práctico es utilizar una aplicación ya existente. Recordemos que la mayor parte de programas utilizados hoy en día usan una arquitectura común (normalmente basada en la arquitectura *COM*), cosa que a grandes rasgos significa que los objetos que utiliza internamente una apli-



cación pueden ser compartidos por otras aplicaciones.

Así sucede con las aplicaciones del paquete ofimático *Office* en todas sus versiones más recientes.

En el siguiente ejemplo veremos cómo utilizamos las librerías de *Office* para que de manera automática podamos enviar un correo electrónico con un mínimo de programación. Para llevarlo a cabo lo primero que deberemos hacer es activar las referencias a las librerías correspondientes.

Esto como ya conocemos, lo hacemos a través del menú **Proyecto->Referencias** y una vez allí buscaremos el apartado *Microsoft Outlook 9 Object Library* o en su defecto la versión que se tenga instalada de *Outlook*.

Seguidamente declararemos en nuestra aplicación una variable de tipo aplicación *Outlook* para así poder acceder a ella:

```
Dim outapp As
  New Outloutappk.Application
```

Referencias - Proyecto1

Referencias disponibles:

Microsoft Office Web Components Wizards
Microsoft OLAP Designer Server Driver
Microsoft OLE DB provider for OLAP Service
Microsoft OLE DB Service Component 1.0 T
✓ Microsoft Outlook 9.0 Object Library
Microsoft Outlook Express 5.0 Type Library
Microsoft PowerPoint 9.0 Object Library
Microsoft Project 8.0 Object Library
Microsoft Remote Data Object 2.0
Microsoft Remote Data Services 2.1 Library
Microsoft Remote Data Services Server 2.1
Microsoft Repository
Microsoft Scripting Runtime
Microsoft Scriptlet Library

Aceptar
Cancelar
Examinar...
Prioridad
Ayuda

Microsoft Outlook 9.0 Object Library
Ubicación: C:\ARCHIVOS DE PROGRAMA\MICROSOFT OFFICE 2000\OFF
Idioma: Estándar

Figura 5.- Activando el apartado de *Microsoft Internet Controls*.

Y ahora que ya tenemos referenciada la aplicación podemos acceder con otra variable al objeto interno de correo electrónico:

```
Dim mailout As
  Outloutappk.MailItem
```

Veamos ahora la rutina que permite enviar correo:

```
Private Sub
    CommandButton1_Click()
  Set mailout =
    outapp.CreateItem(olMailItem)
  mailout.To =
    "solop@virtualsw.es"
  mailout.Subject = "Título del
    mensaje"
  mailout.Attachments.Add ruta
  mailout.Display
End Sub
```

Lo primero que hacemos en ella es crear un nuevo elemento de correo con el método *CreateItem* del objeto *Mail* de la aplicación *Outlook*. Una vez creado tan sólo deberemos llenar las diferentes propiedades de este objeto, que se indican a continuación:

- *To* para indicar a quién se dirige el mensaje.
- *Subject* indicará el cuerpo del mensaje o título del mismo.
- También podemos utilizar el método *Add* del objeto *Attachments* donde le indicamos la ruta del archivo que deseamos adjuntar.

Esta línea la repetiremos tantas veces como archivos deseemos incluir en nuestro correo. Y finalmente con el método *Display* mostraremos al usuario el mensaje.

Quedan en el tintero, pues no es el objetivo de este artículo, muchas propiedades y muchos métodos para acceder a los objetos de *Outlook*. Tan sólo comentar que los objetos, junto con sus propiedades y métodos los podemos encontrar siempre a través del examinador de objetos que proporciona *Visual Basic* (ver Figura 4).

INTERNET EXPLORER

CÓMO EJECUTARLO A PANTALLA COMPLETA

Para ejecutar *Internet Explorer* a pantalla completa lo primero que deberemos hacer es declarar una variable que haga referencia al objeto de vista de documento de *Internet Explorer*:

```
Dim objIE4 As
   SHDocVw.InternetExplorer
```

Aunque para realizar lo anterior deberemos asegurarnos en primer lugar de que en el apartado **Referencias** del menú **Proyecto** (ver gráfico) tengamos activas las referencias a la librería de enlace dinámico **Shodocvw.dll**. Esto lo lograremos activando la casilla correspondiente a *Microsoft Internet Controls*.

Seguidamente inicializaremos la variable con el objeto que tenemos referenciado, en la que modificaremos la propiedad *Theater Mode* que es la que indica si *Internet Explorer* debe inicializarse a pantalla completa o no.

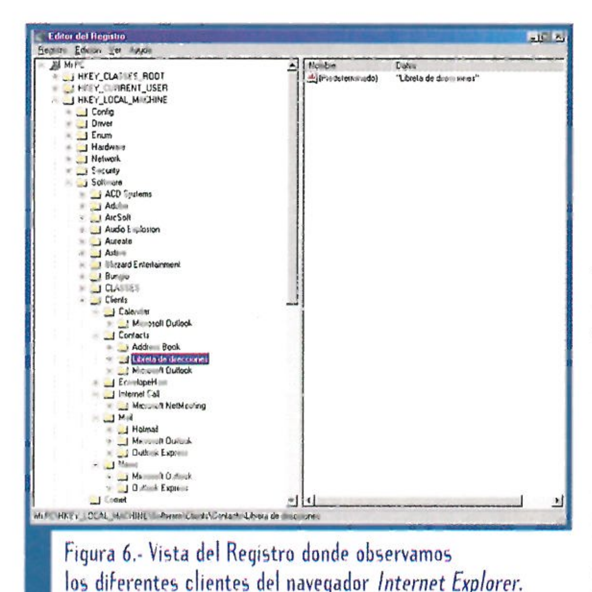


Figura 6.- Vista del Registro donde observamos los diferentes clientes del navegador *Internet Explorer.*

Al mismo tiempo le asignamos una dirección inicial:

```
Public Sub IE_Full_screen_mode()
    Dim objIE4 As
    SHDocVw.InternetExplorer
    Set objIE4 = New
    SHDocVw.InternetExplorer
    With objIE4
        .TheaterMode = True
        .Navigate "www.yahoo.com"
    End With
End Sub
```

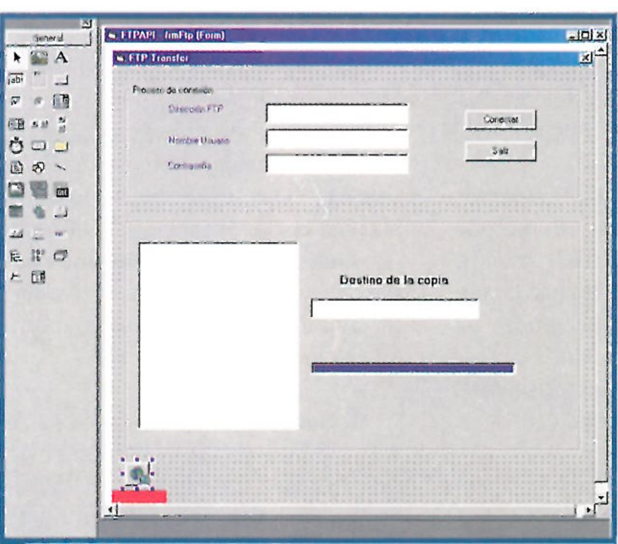


Figura 7.- Detalle del Registro donde vemos cómo acceder a la ruta de los distintos clientes.

LOS CLIENTES DE INTERNET EXPLORER

Vamos a ver ahora cómo podemos obtener los diferentes programas clientes de *Internet Explorer*, es decir los programas que *Internet Explorer* utiliza para ejecutar funciones distintas a las propias del navegador. En este apartado se engloban por ejemplo el correo electrónico, las noticias, los contactos, la libreta de direcciones y las conversaciones o videoconferencias.

Para ello lo que deberemos hacer es leer el Registro de *Windows 9x*, que actúa como una inmensa base de datos donde el sistema guarda todas sus configuraciones y las de las aplicaciones que se ejecutan en él.

Los datos que devolverá esta rutina una vez ejecutada serán las rutas en el disco duro donde se encuentran los programas clientes. Podemos observar de forma manual dichas rutas, por ejemplo si vamos al examinador del Registro mediante el botón **Inicio>Ejecutar** y una vez allí escribimos que ejecute el programa **Regedit.exe**.

Una vez dentro del programa buscaremos en el árbol la carpeta *HKEY_-LOCAL_MACHINE* y en ella localizaremos la subcarpeta *Software* y seguidamente *Clients*.

Aquí es donde tenemos la información de los distintos clientes. Y si después de haber encontrado algún cliente vamos a las carpetas **Shell + open + Command** (en cualquier cliente que hayamos elegido) encontraremos allí la ruta del disco duro del programa cliente, que es lo que nuestro programa va a realizar de forma automática.

Lo primero que vamos a necesitar van a ser las rutinas necesarias para acceder al Registro y poder acceder a los valores que deseemos. Para ello en la sección de declaraciones añadiremos las declaraciones correspondientes a las rutinas *RegCloseKey*, *RegOpenKeyEx* y *Reg QueryValueEx*, todas ellas presentes en la rutina *advapi32.dll*.

Si no conocemos la declaración de estas rutinas podemos obtenerlas fácilmente a través del *Visor de API* (ver Figura 8).

Iremos cambiando el valor de la propiedad RemotePort para así cambiar el puerto que se escanea

Dicho visor lo podemos encontrar en el menú **Complementos->Visor de API.** Suponiendo que en él no aparezca esta opción, entonces significará que no lo tenemos cargado en memoria, paso que deberemos hacer antes de iniciar este procedimiento.

Para ello iremos al Menú **Complementos>Administrador de Complementos** y allí marcaremos el *Visor de API* junto con la caja de opción que indica que lo cargue en memoria. De esta forma conseguiremos que al salir del administrador de complementos veamos en el menú complementos el citado visor.



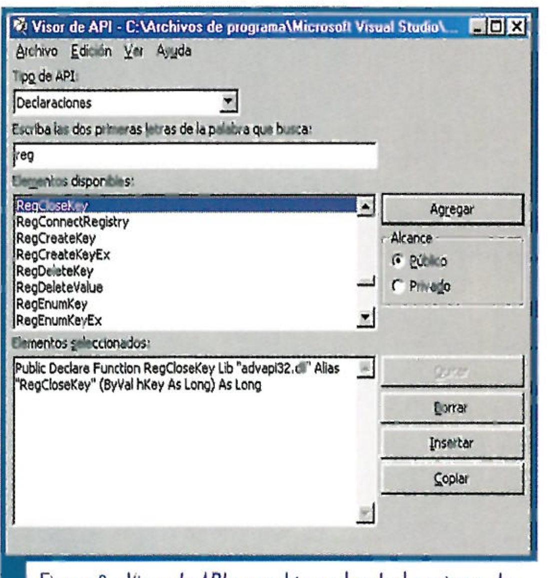


Figura 8.- *Visor de API* para obtener las declaraciones de las funciones de la *API*.

A continuación deberemos declarar una estructura llamada *TiposCliente* que poseerá los nombres de las claves de los programas cliente a los que queremos acceder.

```
Private Type TiposCliente
    Mail As String
    News As String
    Calendar As String
    Contacts As String
    Internet_Call As String
End Type
```

Vamos ahora a examinar la rutina principal de esta pequeña aplicación:

```
Private Function
    ObtenerClientes() As
    TiposCliente
  Static strNombreLlave As
    String,
  strClaves(5) As String,
  byttmp As Byte,
  strT1 As String

  strClaves(1) = "Mail"
  strClaves(2) = "News"
  strClaves(3) = "Calendar"
  strClaves(4) = "Contacts"
  strClaves(5) = "Internet Call"
    For byttmp = 1 To 5
        strNombreLlave =
"Software\Clients\" +
strClaves(byttmp) + "\"
        strT1 =
GetReg(&H80000002,
strNombreLlave, "")
        strNombreLlave =
strNombreLlave + strT1 +
"\Shell\Open\Command\"
        strT1 =
GetReg(&H80000002,
strNombreLlave, "")
        strClaves(byttmp)
= strT1
    Next byttmp

    ObtenerClientes.Mail
 = strClaves(1)
    ObtenerClientes.News
  = strClaves(2)
  ObtenerClientes.Calendar =
   strClaves(3)
  ObtenerClientes.Contacts =
   strClaves(4)
  ObtenerClientes.Internet_Call
   = strClaves(5)
End Function
```

En ella lo primero que hacemos es llenar un *array* de cadena con los nombres de las claves que vamos a encontrar en el registro para los distintos clientes, es decir el nombre de la clave *Mail* para el programa cliente de *Mail*, *news* para el cliente de lector de noticias, *Calendar* para la agenda o calendario, *contacts* para la libreta de direcciones de los contactos e *Internet call* para el programa de videoconferencia.

Posteriormente a través de un bucle iremos obteniendo los diferentes valores de las claves donde, tal y como hemos visto antes, se guardan las diferentes rutas de acceso al disco duro para cada cliente elegido. Una vez que hayamos obtenido dicha información la insertaremos en la estructura *strClaves*.

El usuario puede acceder a los objetos internos de la aplicación Office

Finalmente pondremos en la estructura global *ObtenerClientes* los valores que hemos obtenido y que se encuentran grabados en la estructura de tipo cadena *strClaves*.

CONCLUSIÓN

Con este artículo hemos visto varias rutinas globales que aunque no se encuentran incluidas dentro de los apartados de esta serie, sí que los complementan de forma muy útil, permitiéndonos así crear una *suite* completa para el acceso a *Internet* mediante la herramienta de desarrollo *Visual Basic*.

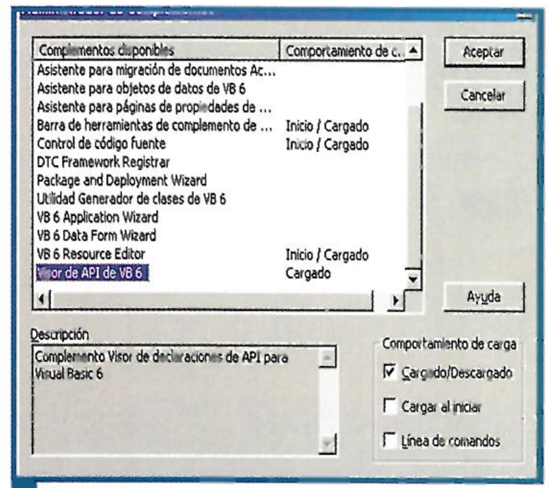
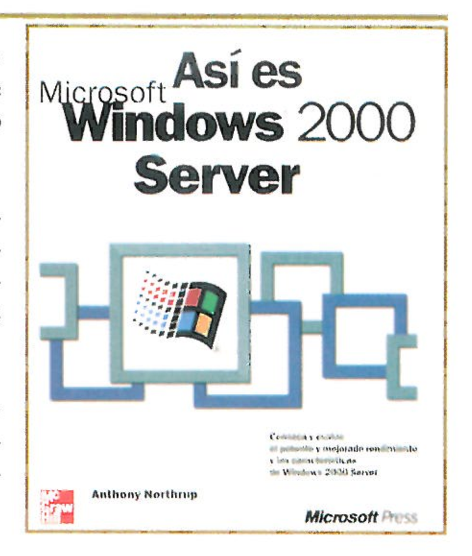


Figura 9.- Cargando en el entorno de desarrollo de *Visual Basic* el complemento *Visor de API*.



ASÍ ES MICROSOFT WINDOWS 2000 SERVER

Adelántese a sus competidores con esta primera revisión oficial de *Microsoft Windows* 2000 *Server,* información esencial para los profesionales de las *TI* que necesitan evaluar y prepararse para este sistema operativo empresarial de siguiente generación.



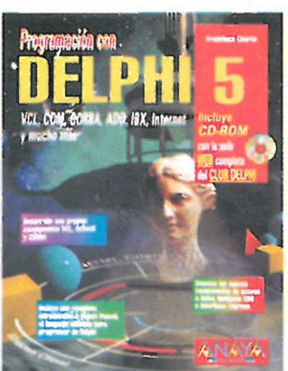
Este manual proporciona una visión técnica autorizada de las características clave del servidor, con profundización en los servicios de red esenciales. Obtendrá una valiosa información para realizar la evaluación y planificación de sistema, e ideas avanzadas que le pueden ayudar a simplificar sus tareas y reducir los costes.

Con este libro se introducirá en las posibilidades de *Windows* 2000 *Server* con una evaluación de las mejoras del sistema, incluyendo un administrador de red más fácil, soporte avanzado de computación distribuida y gestión empresarial basada en *Web*.

Editorial: Mc Graw Hill
Nº de Páginas: 201
Nivel: Inicial
Autor: Tony Northrunp
Idioma: Español
Precio: 2.500 Ptas. (I.V.A. inc.)

PROGRAMACIÓN CON DELPHI 5





Borland *Delphi* sigue evolucionando y con su nueva versión (v. 5) lo hace convirtiéndose en una herramienta de última generación sumamente avanzada. Este libro pretende servir al lector como elemento imprescindible para poder aprender a desarrollar aplicaciones con *Delphi* 5.

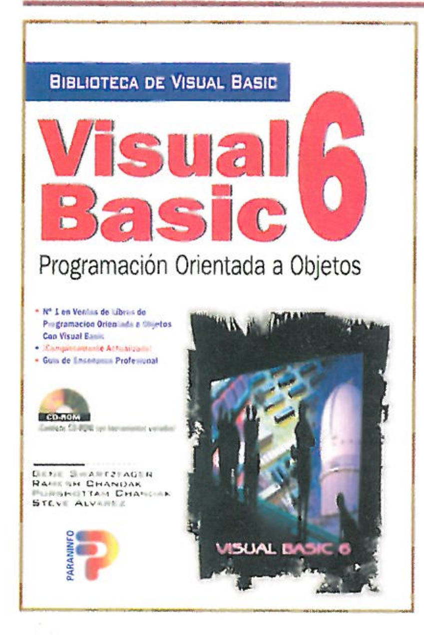
En el manual encontrará una definición clara de la estructura general del lenguaje *Object Pascal*, de muchos de los componentes *Delphi* y su utilización, así como de los distintos aspectos de la programación sobre *Windows*, etc.

El *CD-ROM* que acompaña al libro incluye los proyectos y código fuente de los ejemplos y además la sede completa del *Web* Club *Delphi* para acceder a todos los recursos, trucos, componentes, etc, sin tener que conectarse a la Red.

Entre otros contenidos cabe destacar algunos como: la manipulación de archivos, la comunicación entre aplicaciones, el uso de la *API* de *Windows*, los fundamentos del tratamiento de datos, o la mejora de las interfaces entre otros.

Editorial: Anaya Multimedia
Nº de Páginas: 1.000
Nivel: Avanzado
Autor: Francisco Charte Ojeda
Idioma: Español
Precio: 6.995 Ptas. (I.V.A. inc.)

VISUAL BASIC 6 PROGRAMACIÓN ORIENTADA A OBJETOS



Visual Basic 6 y la programación orientada a objetos (*POO*) forman una combinación imbatible, y este libro muestra cómo dominar el arte de escribir aplicaciones orientadas a objetos con una calidad a nivel profesional.

En esta detallada guía aprenderá, paso a paso, a crear *software* realmente reutilizable para obtener el máximo provecho con el mínimo esfuerzo.

Este libro realiza un profundo estudio a nivel profesional de las técnicas de la *POO*, incluyendo la codificación con objetos, el encapsulado, la creación de aplicaciones con controles *ActiveX*, las bibliotecas de clases y muchos elementos más.

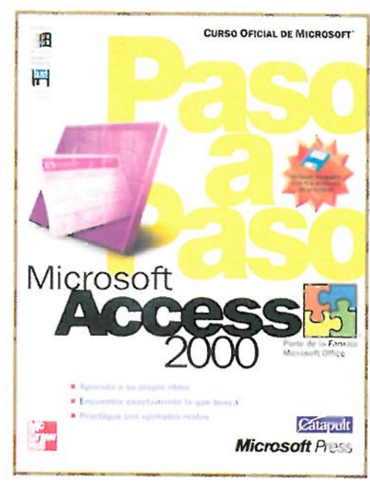
Se trata de un manual especialmente indicado para los programadores y desarrolladores que deseen mejorar sus habilidades hasta el máximo nivel, así como los que quieran dominar el arte de crear *software* reutilizable orientado a objetos. Con él trabajará con las mejores técnicas de codificación y diseño ahorrándose tiempo, creará clases, métodos y aplicaciones potentes y flexibles que podrá utilizar continuamente en sus aplicaciones o creará componentes *COM* entre otras muchas posibilidades.

Editorial: Paranınfo
Nº de Páginas: 454
Nivel: Avanzado

Autores: Swartzfager y otros
Idioma: Español
Precio: 4.400 Ptas. (I.V.A. inc.)

PASO A PASO MICROSOFT ACCESS 2000

Este libro se presenta como la forma más rápida y fácil de aprender a utilizar la última versión de la popular base de datos de *Microsoft*. Gracias a él será capaz de instalar y visualizar información, añadir y editar datos utilizando formularios y tablas, ordenar y filtrar registros, previsualizar e imprimir informes y crear etiquetas de correo.

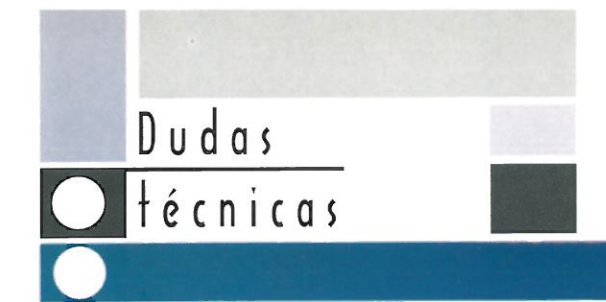


Además aprenderá a crear y ampliar bases de datos, relacionar tablas, trabajar con datos externos, convertir los datos en información significativa, diseñar consultas y análisis de datos y combinar la información de dos tablas en un único formulario.

Entre estas páginas descubrirá algunos consejos que le proporcionarán pasos alternativos e información adicional en su aprendizaje, así como ejemplos reales que le servirán a la hora de realizar sus prácticas.

Editorial: McGraw Hill
Nº de Páginas: 344
Nivel: Inicial

Autor: Catapult
Idioma: Español
Precio: 4.995 Ptas. (I.V.A. inc.)



Dudas técnicas

En esta sección, como cada mes, SÓLO PROGRAMADORES os brinda la oportunidad de encontrar respuesta a las dudas que podáis tener, en cualquier tema relacionado con la programación y bajo cualquier entorno de desarrollo. Ya sabéis que nuestra dirección es solop@virtualsw.es

PREGUNTA

Amigos de *Sólo Programadores:*

Estoy aprendiendo a realizar las páginas *ASP* en el curso de vuestra revista, pero me encuentro en un apuro. Mediante el programa *Cute FTP* he llevado la página a mi *Web*, pero una vez allí no funciona bien. Aparece la página *ASP* -página que dice la fecha del servidor- igual que si la ejecutara directamente con *Internet Explorer* en mi ordenador, y no como se ve ejecutando el *PWS*.

Estoy aprendiendo y me preguntaba si existe algo más en la trasferencia del archivo -lo he traspasado en *ASCII*- o cualquier otra cosa.

También he escrito al *Webmaster* para ver si me puede aclarar el problema. Además tengo un pequeño *script* en *ASP* que me dejaron, que cuando intento acceder a él me dice que si quiero descargarme el archivo, como si fuera de *FTP*.

Agradezco su tiempo y esfuerzo.

RESPUESTA

Estimado lector:

Por lo que deducimos de tu consulta todavía no has entendido por completo la diferencia entre las páginas *HTML* normales y corrientes, y las páginas *ASP*. No te preocupes, intentaremos incidir en ello con el fin de que te quede todo claro y sepas qué es lo que puedes hacer y qué no para desarrollar tu sitio *Web* y sacar partido a ese espacio que te concede tu servidor.

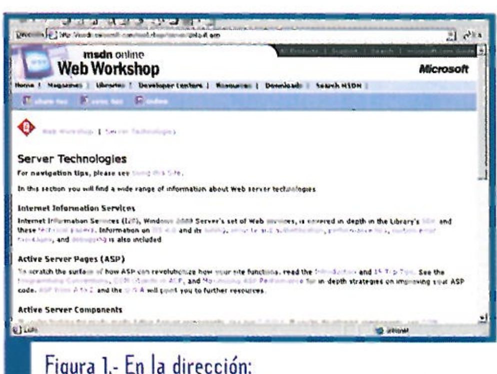


Figura 1.- En la dirección: *msdn.microsoft.com/workshop/server/default.asp* podemos encontrar abundante información en torno a la tecnología *ASP*.

Cuando tú estás navegando por *Internet* y sigues un enlace que te lleva a otra página *HTML*, el servidor donde se encuentra esa página recibe una petición. En ese momento lee la página *HTML* y simplemente la envía a tu navegador. Entonces, una vez que está en tu ordenador, el navegador se encarga de interpretar el lenguaje *HTML*, los *scripts* de *Javascript* o incluso los *applets* o los objetos *ActiveX* que puedan ser referenciados dentro de esa página *HTML*. Todo esto sucede en tu ordenador, el servidor terminó su tarea con enviarte el código de la página.

Por su parte, las páginas *ASP* adoptan un proceso ligeramente distinto. Cuando tú sigues un enlace que conduce a una página *ASP* el servidor no se limita a mandar al navegador el contenido de la misma, sino que interpreta la página y el resultado de esa interpretación es lo que llega al navegador del usuario final. El código contenido en las páginas *ASP* es una mezcla de *HTML* y código propio de las aplicaciones *ASP*. Con el primero el servidor no hace nada, pero el segundo ha de



Figura 2.- *The Web Developer's Virtual Library*, www.stars.com/WDVL, es un excelente sitio *Web* donde aprender todo lo relacionado con las distintas tecnologías de *Internet*.

ejecutarlo. Por lo tanto, para utilizar páginas *ASP* debes estar seguro de que el servidor correspondiente las soporte.

Cuando un servidor recibe una petición que hace referencia a un tipo de archivo que desconoce, o que conociéndolo no lo considera como una página *HTML* ni como un programa ejecutable por él, lo que hace es enviarlo al navegador igual que si se tratara de un archivo, como cuando haces clic en un enlace que señala al un fichero *ZIP*, por ejemplo. Esta es la razón por la que al hacer clic en tus páginas *ASP* el navegador abre la ventana de descarga de ficheros.

PREGUNTA

Un saludo:

Debido al auge que ha experimentado *Internet* he intentado introducirme un poquito en estas tecnologías pero he sufrido bastante ante la inmensa información que existe y no he sido capaz de filtrarla ya sea a través de libros o de publicaciones para mis propósitos específicos.

Quizá mi duda sea sencilla pero sólo necesito crear una página *Web* capaz de realizar consultas sobre una base de datos, que tendré que poner en el servidor. La duda surgió ante la elección de *JScript, VBScript, CGI, Perl*, etc. Vamos que no me aclaraba.

Por lo que he ido entendiendo la cuestión radica en generar un fichero de consulta *ASP*, con el lenguaje que sea, por ejemplo el *JScript* que se ha explicado en el número 60 de *Sólo Programadores*, y ponerlo en el servidor. Pero mi problema es sencillo: no encuentro el servidor que me permita poner mi página *ASP* (por lo visto que soporte *IIS*). Así mi página que tenia en principio sólo *HTML* estático ubicada en **www.geocities.com** no me permite ahora incluir los ficheros *.MDB* ni *.ASP*.

Marcos González.
Las Palmas de Gran Canaria.

RESPUESTA

Estimado amigo:

Internet ha crecido tanto y en tan poco tiempo que las dudas que nos expones en tu mensaje son frecuentes. La mayoría de las personas que se inician en la programación de aplicaciones tienen esas preguntas y su respuesta en profundidad rebasa sobradamente los límites de esta sección. Sin embargo vamos a tratar de resumir aquí las claves más importantes que te ayuden a diferenciar entre unas y otras tecnologías, y que en definitiva te orienten en la búsqueda de la solución que mejor se ajuste a lo que deseas.

Internet es un medio basado en una arquitectura cliente/servidor. De forma resumida podríamos decir que los navegadores de los clientes son los usuarios que realizan peticiones a los servidores, y éstos se encargan de servirlas transmitiendo las páginas *HTML* que luego el usuario ve.

Algunas tecnologías están creadas para los servidores y otras para los clientes. Esta dicotomía es cada vez más confusa en sentido estricto, pero por el momento puedes considerarlo así. Por ejemplo, el lenguaje *JavaScript* o el lenguaje *VBScript* fueron concebidos para incorporar pequeños *scripts* en las páginas *HTML*. Éstos se ejecutan en el navegador del cliente cuando la página es cargada. Los *applets* funcionan de la misma forma.

En el lado del servidor podemos encontrar programas *CGI* hechos en *C* o en *Perl*, por ejemplo. Son ejecutados por el servidor y su respuesta es enviada al cliente.

Las páginas *ASP* son un sistema distinto. Se trata de páginas *HTML* en las que se ha intercalado código que ha de ejecutar el servidor. Este código puede ser *JScript* o *VBScript* -por lo que podemos decir que con esta tecnología esos lenguajes han dado el salto del cliente al servidor-. El resultado, la página *HTML* más la salida del código ejecutado, se envía al navegador del cliente.

Cuando en este esquema se introducen las bases de datos siempre estamos hablando del lado del servidor. Es decir, la base de datos es un recurso al que se accede en general por medio de programas que se ejecutan en el servidor. Las soluciones que puedes utilizar son inumerables pero quizás lo más importante es que primero hables con el administrador de la máquina donde se van a situar tus páginas *Web* para conocer con mayor exactitud cuáles son las posibilidades con

las que cuentas. En tu caso con el administrador de *Geocities*.

Hoy en día no es frecuente encontrar servidores que sean capaces de soportar páginas *ASP*. Especialmente si estamos hablando de soluciones no profesionales, es decir, del espacio gratuito que casi todos los proveedores de servicios ofrecen a sus clientes para colocar sus páginas *Web*. Lo que si que es posible que encuentres son servidores donde te permitan colocar algunos *CGI* hechos en *Perl* o en *C*. Esta solución limita considerablemente tus opciones ya que el uso de base de datos de datos como tal debe ser descartada.

Si tu base de datos es relativamente sencilla y no demasiado grande, puedes plantearte la posibilidad de transformarla de alguna forma en ficheros de texto y utilizar programas *CGI* hechos *Perl* para acceder a ella. Esta solución no está desde luego exenta de problemas pero, eliminado las alternativas profesionales y de pago, es casi seguro la que mejor vas a poder aplicar. Además *Perl* cuenta con una ventaja adicional: pertenece al sistema *Unix* y la mayoría de los servidores son *Unix*, pero al mismo tiempo también se puede utilizar en *Windows*. Al ser un lenguaje interpretado y no compilado, puedes hacer las pruebas que precises con tus programas en tu ordenador personal y luego subir los *scripts* al servidor y simplemente cambiar los datos relativos a las rutas absolutas de los ficheros. *Perl* es un excelente lenguaje para el tratamiento de cadenas de texto y no resulta demasiado difícil de aprender.

Otra posible solución que puedes contemplar es la que consiste en transformar tu base de datos en archivos *XML* y aprovechar las ventajas de los navegadores para tratar este tipo de ficheros como fuentes de datos. Esta alternativa tiene la desventaja de que hoy en día por el momento sólo *Internet Explorer* 5.x soporta de manera natural el estándar *XML*, con lo que estarías limitando considerablemente la audiencia de tu sito *Web*.

PREGUNTA

Estimados amigos:

En primer lugar os quiero felicitar por vuestro trabajo, ya que publicáis una revista muy elaborada e interesante. Y después del saludo de rigor, ahí va mi pregunta.

Estoy trabajando con *Visual Basic* 6.0 contra una base de datos *Access* 97. Voy a distribuir mi aplicación, pero quiero evitar que los clientes puedan ver el contenido de la base de datos (no sólo no poder tocarla, sino ni siquiera ver los objetos). Si fuera otro gestor podría hacerlo con usuario/contraseña vía *ODBC*, pero al ser *Access* creo que esto no es posible. A ver si me podéis echar un cable.

Un saludo y gracias de antemano.

Jerónimo Ballaz. Pamplona.

RESPUESTA

Estimado lector:

Access es bastante limitado si queremos utilizarlo a modo de sistema gestor de bases de datos. En estos casos se suele recurrir a herramientas mucho más potentes y preparadas para entornos cliente/servidor como son *Oracle* o *Microsoft SQL Server*.

En cualquier caso en la propia documentación de *Access* podrás encontrar diversa información en-

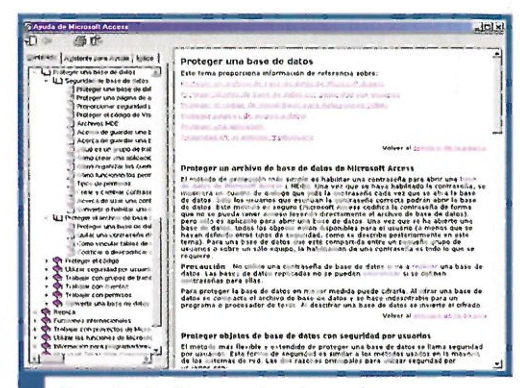


Figura 3.- Ayuda proporcionada por *Access* acerca de cómo podemos proteger nuestra base de datos.

torno a cómo puedes incorporar niveles de seguridad a tu base de datos. Entre las garantías que puedes aplicar se encuentran:

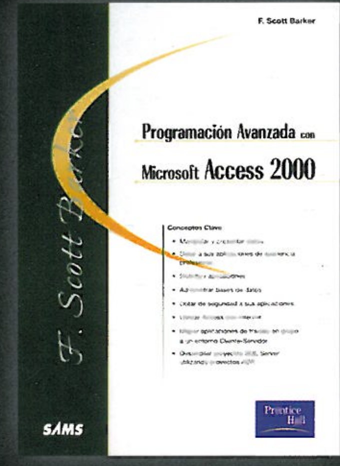
- Proteger una base de datos con contraseña. El inconveniente principal que presenta esta solución es que los usuarios podrán tener acceso total a la base de datos toda vez que hayan introducido la contraseña.

- Proteger una base de datos estableciendo perfiles de usuarios. *Access* define por defecto dos perfiles de usuario: el administrador y todos los demás. Como es evidente el administrador tendrá la capacidad de hacer cualquier cosa. El resto de los usuarios no podrán modificar tablas, etc. El número de perfiles pueden modificarse convenientemente.

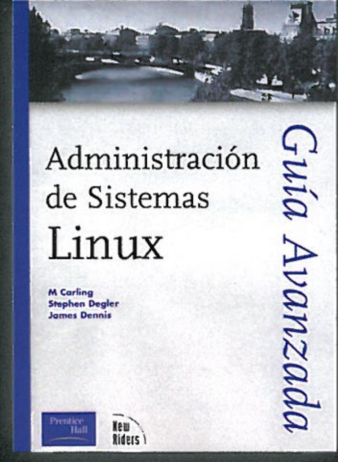
Te recomendamos que experimentes con estos y otros mecanismos propuestos por *Access* con el fin de dar con el sistema de seguridad que más se ajuste a tus necesidades, e incluso que consultes las ventajas que proporciona la nueva versión de *Access*, *Access* 2000, al respecto. En cualquier caso, si ninguno de ellos te satisface o cubre plenamente tus requisitos quizás sería el momento de pasar a utilizar un gestor de base de datos más apropiado para el tipo de entorno del que nos hablas en tu consulta.

Novedades para profesionales escritos por profesionales



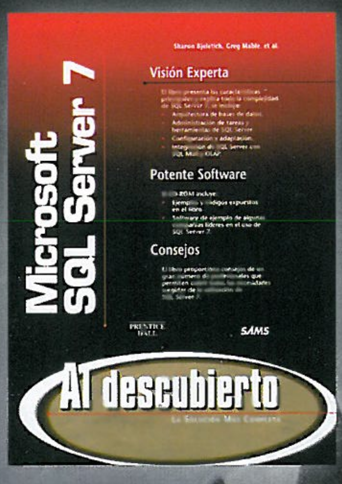


Programación Avanzada con Microsoft Access 2000
F. Scott Barker



Administración de Sistemas Linux
M Carling
Stephen Degler
James Dennis





Microsoft SQL Server 7 Al Descubierto
Sharon Bjeletich et al.

Incluye CD-Rom



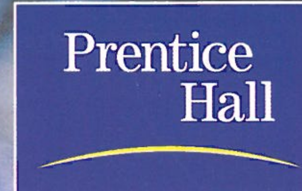
Edición Especial TCP/IP
John Ray



Incluye CD-Rom

Manual de XML
Charles F. Goldfarb & Paul Prescod

¡Ya a la venta en tu librería habitual!



Líder mundial en libros y recursos de informática